

AFRL-IF-RS-TR-2001-235
Final Technical Report
November 2001



INTEGRATED DESIGN ENVIRONMENT FOR ASSURANCE (IDEA)

Harris Corporation

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J759/00

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

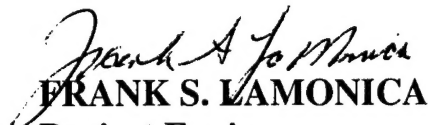
20020308 061

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-235 has been reviewed and is approved for publication.

APPROVED:


FRANK S. LAMONICA
Project Engineer



FOR THE DIRECTOR:

MICHAEL L. TALBERT, Major, USAF
Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

INTEGRATED DESIGN ENVIRONMENT FOR ASSURANCE (IDEA)

Joe Tallet

Contractor: Harris Corporation

Contract Number: F30602-00-C-0067

Effective Date of Contract: 20 April 2000

Contract Expiration Date: 17 July 2001

Short Title of Work: Integrated Design Environment for Assurance
(IDEA)

Period of Work Covered: Apr 00 – Jul 01

Principal Investigator: Joe Tallet

Phone: (321) 984-6376

AFRL Project Engineer: Frank S. Lamonica

Phone: (315) 330-2055

Approved for public release; distribution unlimited.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Frank S. Lamonica, AFRL/IFTD, 525 Brooks Rd, Rome, NY, 13441-4505.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE Nov 01	3. REPORT TYPE AND DATES COVERED Final Apr 00 - Jul 01		
4. TITLE AND SUBTITLE INTEGRATED DESIGN ENVIRONMENT FOR ASSURANCE (IDEA)		5. FUNDING NUMBERS C - F30602-00-C-0067 PE - 63760E PR - IAST TA - 00 WU - 05		
6. AUTHOR(S) Joe Tallet				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Harris Corporation Government Communications Systems Division PO Box 37 Melbourne, FL 32902		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington, VA 22203-1714		10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2001-235		
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Frank S. Lamonica, IFTD, 315-330-2055				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) This report covers the research and development (R&D) performed by Harris Corporation under a DARPA sponsored effort entitled Integrated Design Environment for Assurance (IDEA). The intent of IDEA was to provide Information Assurance (IA) analysts with the capability to select appropriate software IA tools, seamlessly link them to system data and other tools, initiate and monitor execution, and merge and present results. A key feature is the incorporation of Fuzzy Fusion technology to enable the merging (fusion) of results from individual tools and the presentation of those results in a composite fashion to provide a concise and timely report of a system's security posture. IDEA was performed under DARPA's Information Assurance Science & Engineering Tools (IASET) program. The IASET program came to a premature conclusion in FY00. As a result, Harris Corporation was not able to develop a demonstrable prototype as originally planned. An architectural design, however, was completed and is reported on.				
14. SUBJECT TERMS Information Assurance, Software Tools, Tool Integration, Environments, Data Fusion, Common System Model		15. NUMBER OF PAGES 296		
		16. PRICE CODE		
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

1	SCOPE	1
1.1	ORIGINAL EFFORT	1
1.2	INITIAL FUZZYFUSION™ CAPABILITY	3
1.3	FUZZYFUSION™ RESULTS	6
2	DOCUMENT OVERVIEW	7
3	REFERENCE DOCUMENTS	8
4	PROGRAM SUMMARY	9
4.1	LEVELS OF DATA FUSION	9
4.2	OPERATIONAL CONCEPT FOR A FUZZYFUSION™ COMPONENT	11
4.2.1	Introduction	11
4.2.2	Distributed Real-Time Vulnerability Measurement	12
4.2.3	Level 1 FuzzyFusion™ - What is the vulnerability of each node?	12
4.2.4	Level 2 FuzzyFusion™ - How do vulnerabilities in one node affect other nodes?	13
4.2.5	Level 3 FuzzyFusion™ - How do I get to a vulnerable node that allows me access to critical data?	16
4.2.6	Level 4 FuzzyFusion™ - How do I increase the level of difficulty of following a path to the critical node?	19
4.2.7	Summary	19
4.3	TECHNICAL APPROACH	20
4.3.1	The FuzzyFusion™ component database	20
4.3.2	Client vulnerability mapping guidelines	21
4.3.3	The FuzzyFusion™ component	24
4.3.4	Data input requirements for the component	25
4.3.5	Processing the data	27
4.4	THE PROOF-OF-CONCEPT PROTOTYPE	34
4.4.1	LOGICAL ARCHITECTURE	34
4.5	TESTING AND EVALUATION	51
4.5.1	Analysis of results	51
4.5.2	Quality of Results	51
4.5.3	Comparison of results	53
4.6	RESEARCH INTO NATURAL LANGUAGE PROCESSING	58
4.6.1	Discussion of Research	58
5	CONCLUSIONS	61
5.1	IDEA PROGRAM CONCLUSIONS	61
5.2	AREAS FOR IMPROVEMENT	61
6	FUTURE WORK	62
6.1	REAL-TIME VULNERABILITY ANALYSIS	62
6.2	REAL-TIME INTRUSION DETECTION	62
6.3	FIXING VULNERABILITIES	62
6.4	MAINTAINING HISTORIES	63
6.4.1	Trust	63
6.5	SECURITY POLICY META-LANGUAGE	63
6.6	GENERALIZATION OF FUZZY FUSION™	64
6.7	NATURAL LANGUAGE PROCESSING OF VULNERABILITIES	64
7	APPENDIX A – DATA	65
7.1	IVE TABLE	65
7.2	CONDITION TABLE	77

7.3	RULE TABLE.....	77
7.4	CVE INFORMATION.....	88
7.5	FUZZY SET CONTRIBUTIONS.....	93
8	<u>APPENDIX B – XML INFORMATION</u>	95
8.1	<u>XML – A BETTER COMPONENT PROTOCOL</u>	95
8.2	<u>XML</u>	97
8.2.2	<i>Dissimilar Component Models</i>	99
8.2.3	<i>Lack of ORB Support</i>	99
8.2.4	<i>Interface Clashes</i>	100
9	<u>APPENDIX C– PROTOTYPE INSTRUCTIONS</u>	104
9.1	<u>USING DTS TO IMPORT SCANNER TEST DATA FROM AN ACCESS DATABASE INTO AN MSDE DATABASE</u>	104
9.2	<u>IDEA XML GENERATOR</u>	111
9.3	<u>IDEA DEMO INTERFACE</u>	114
10	<u>APPENDIX D – IDEA ARCHITECTURE</u>	116
11	<u>APPENDIX E – META-LANGUAGE</u>	264
12	<u>APPENDIX F – IDEA: AN INFORMATION SUPERSTRUCTURE</u>	272

List of Figures

FIGURE 1.1-1 – INITIAL SCOPE OF IDEA AS PRESENTED AT THE 2 ND BLUE RIBBON SESSION	2
FIGURE 1.2-1 – EARLY VIEW OF FUZZYFUSION™	4
FIGURE 1.2-2 – THE FUZZYFUSION™ COMPONENT SUPPORTING HIGHER LEVELS OF DATA FUSION	4
FIGURE 1.2-3 – DATA FUSION LEVELS AS PRESENTED IN THE 3 RD BLUE-RIBBON SESSION	5
FIGURE 1.2-4 – FUZZYFUSION™ AS A COMPONENT TECHNOLOGY	5
FIGURE 4.2-1 – XML IS THE DATA TRANSPORT MECHANISM FOR THE FUZZYFUSION™ COMPONENT	11
FIGURE 4.2-2 – THE HOST APPLICATION USES A VULNERABILITY SCANNER TO GENERATE DATA FOR THE FUZZYFUSION™ COMPONENT	11
FIGURE 4.2-3 – DISTRIBUTED REAL-TIME VULNERABILITY ASSESSMENT APPLICATION	12
FIGURE 4.2-4 – NETWORK CONFIGURATION	13
FIGURE 4.2-5 – VULNERABILITY ASSESSORS LINKED TOGETHER	14
FIGURE 4.2-6 – EXPLOITATION PATH THROUGH A NETWORK	15
FIGURE 4.2-7 – DEPENDENT NODE, NODE 52 RAISES ITS VULNERABILITY LEVEL	15
FIGURE 4.2-8 – KEY NODE ON NETWORK BECOMES MORE VULNERABLE	16
FIGURE 4.2-9 – EXTERNAL TO INTERNAL ATTACK PATH TO A KEY NODE	18
FIGURE 4.2-10 – INTERNAL TO INTERNAL ATTACK PATH TO KEY NODE	18
FIGURE 4.3-1 – DATA FOR THE FUZZYFUSION™ COMPONENT	20
FIGURE 4.3-2 – MAPPING TOOL VULNERABILITIES TO IVEs	22
FIGURE 4.3-3 – PRUNING FORMULA	27
FIGURE 4.3-4 – INTERMEDIATE CONDITION FORMULA	27
FIGURE 4.3-5 – DIAGRAM OF FUZZYFUSION™ INPUTS	28
FIGURE 4.3-6 – DIAGRAM OF FUZZYFUSION™ OUTPUTS	29
FIGURE 4.3-7 – CALCULATION OF NUMERIC GRADE, SUB-STEP A	30
FIGURE 4.3-8 – CALCULATION OF NUMERIC GRADE, SUB-STEP B	30
FIGURE 4.3-9 – CALCULATION OF BELIEF	30
FIGURE 4.3-10 – CALCULATION OF OPPORTUNITY FOR EXPLOITATION	31
FIGURE 4.3-11 – FUZZY MEMBERSHIP FUNCTION	31
FIGURE 4.3-12 – CHARACTERISTIC MAPPINGS OF 16 INTERMEDIATE CONDITIONS TO END CONDITION FUZZY SETS	31
FIGURE 4.3-13 – FUZZY MEMBERSHIP FUNCTION INCORPORATING FUZZY BELIEF MEASURES	32
FIGURE 4.3-14 – DEFUZZIFICATION METHOD	32
FIGURE 4.3-15 – CALCULATION OF OFE FOR END CONDITION	32
FIGURE 4.3-16 – CALCULATION OF FINAL NODE VALUE	32
FIGURE 4.3-17 – STAT® ANALYZER FUZZYFUSION™ INFERENCING ALGORITHM	33
FIGURE 4.4-1: COMPONENT LAYOUT	34
FIGURE 4.4-2: MAIN	35
FIGURE 4.4-3: MAIN	47
FIGURE 4.4-4: MAIN	48
FIGURE 4.4-5: MAIN	49
FIGURE 4.5-1 – DISTRIBUTION OF SEVERITY LEVELS REPORTED BY STAT® SCANNER DURING FORTY-FOUR SCANS OF TWELVE NODES	53
FIGURE 4.5-2 – FUZZYFUSION™ COMPONENT RESULTS	54
FIGURE 4.5-3 END CONDITIONS USING FUZZY MEMBERSHIP CONTRIBUTIONS AND GRADE AND OFE	55
FIGURE 4.5-4 – MODIFYING OFE	56
FIGURE 4.5-5 – END CONDITIONS USING FUZZY MEMBERSHIP CONTRIBUTIONS AND OFE	57
FIGURE 4.5-6 – MODIFYING OFE	57
FIGURE 6.3-1 – MAPPING OF THE END CONDITIONS TO THE VULNERABILITIES THAT LED TO IT	63
FIGURE 8.1-1 – BASIC MIDDLEWARE ARCHITECTURE	95
FIGURE 8.1-2 – A SIMPLIFIED VIEW	95
FIGURE 8.1-3 – BRIDGE BETWEEN TWO DISSIMILAR COMPONENT MODELS	96
FIGURE 8.1-4 – INTERFACE CONFLICT WITH ADAPTER SOLUTION	96
FIGURE 8.2-1 – AN XML DOCUMENT	98
FIGURE 8.2-2 – XML INTERFACE	98
FIGURE 8.2-3 – XML BRIDGE	99

<u>FIGURE 8.2-4 – XML ADAPTER</u>	<u>100</u>
<u>FIGURE 8.2-5 – XML AGENT.....</u>	<u>100</u>
<u>FIGURE 8.2-6 – EXAMPLE OF IDL</u>	<u>101</u>
<u>FIGURE 8.2-7 – EXAMPLE XML INTERFACE</u>	<u>102</u>
<u>FIGURE 8.2-8 – EXAMPLE XML DOCUMENT</u>	<u>102</u>

List of Tables

TABLE 4.1-1 – THE LEVELS AND PURPOSES OF THE INTELLIGENCE DATA FUSION PROCESS.....	9
TABLE 4.3-1 – IAT MAPPED TO PRE AND POST CONDITIONS.....	21
TABLE 4.3-2 – STAT® SCANNER VULNERABILITY DESCRIPTION	22
TABLE 4.3-3 – IVE AND DESCRIPTION	22
TABLE 4.3-4 – STAT® SCANNER VULNERABILITY INFORMATION.....	23
TABLE 4.3-5 – CVE DESCRIPTION	23
TABLE 4.3-6 – IVE DESCRIPTION	23
TABLE 4.3-7 – MAPPING SUMMARY.....	23
TABLE 4.3-8 – SAMPLE FUZZYFUSION™ COMPONENT OUTPUT (AFTER TRANSLATED FROM XML)	24
TABLE 4.3-9 – TAG HIERARCHY FOR IDEA’S XML DATA.....	26
TABLE 4.6-1 – RESULTS FROM APPLICATION OF FIVE LEARNING ALGORITHMS.	60
TABLE 7.1-1 – IVE INFORMATION	65
TABLE 7.2-1 – VULNERABILITY CATEGORIES AND THEIR DEFINITIONS.....	77
TABLE 7.5-1 – FUZZY MEMBERSHIP FUNCTIONS	93

1 Scope

This document describes the research findings of the Integrated Design Environment for Assurance (IDEA) effort and serves as the final technical report for the effort. It includes the original effort of the IDEA prototype design, as well as additional research activities in the reasoning domain of Fuzzy Fusion™. IDEA was part of the Defense Advanced Research Projects Agency (DARPA) sponsored Information Assurance Science and Engineering Tools (IASET) Program.

1.1 Original Effort

The original IDEA effort's objective was to develop and demonstrate a prototype for an integrated Design Environment for system security engineering/design team members. The objective was to create an information repository, accessible through various Application Programming Interfaces (APIs) and Graphical User Interfaces (GUIs). This repository would contain both inputs to, and outputs of, various information assurance tools, and would serve as the design model for all aspects that are security relevant about a system design. The thought was that tool technology would evolve and mature, but the information generated by a tool, and the information supplied as input, would be treated as data accessible to all tools in a given tool suite. This would preserve the security relevant design information, allow it to be augmented as technology matured, and maintain a living database that represented the state of the system design. Harris developed a detailed design, and delivered the IDEA architecture report as required.

In October 2000, the IASET program was identified by DARPA for early termination. At that time, Harris had fulfilled the six month milestone deliverables for a system design for IDEA. With the termination of IASET, there would be no prototype tools provided by other IASET participants to integrate with the IDEA information repository. In February 2001, it was decided to emphasize FuzzyFusion™ research with remaining funding.

FuzzyFusion™ combines Data Fusion and Fuzzy Logic in a unique way to generate a comprehensible vulnerability assessment result. IDEA was originally scoped to collect results of several external tools and assess the results with the FuzzyFusion™ technology. It was now decided that IDEA would build, test, and deliver a FuzzyFusion™ component that would test the concepts for higher-layers of data fusion technology. This report, in effect is delivering the "Fusing" component illustrated in Figure 1.1-1.

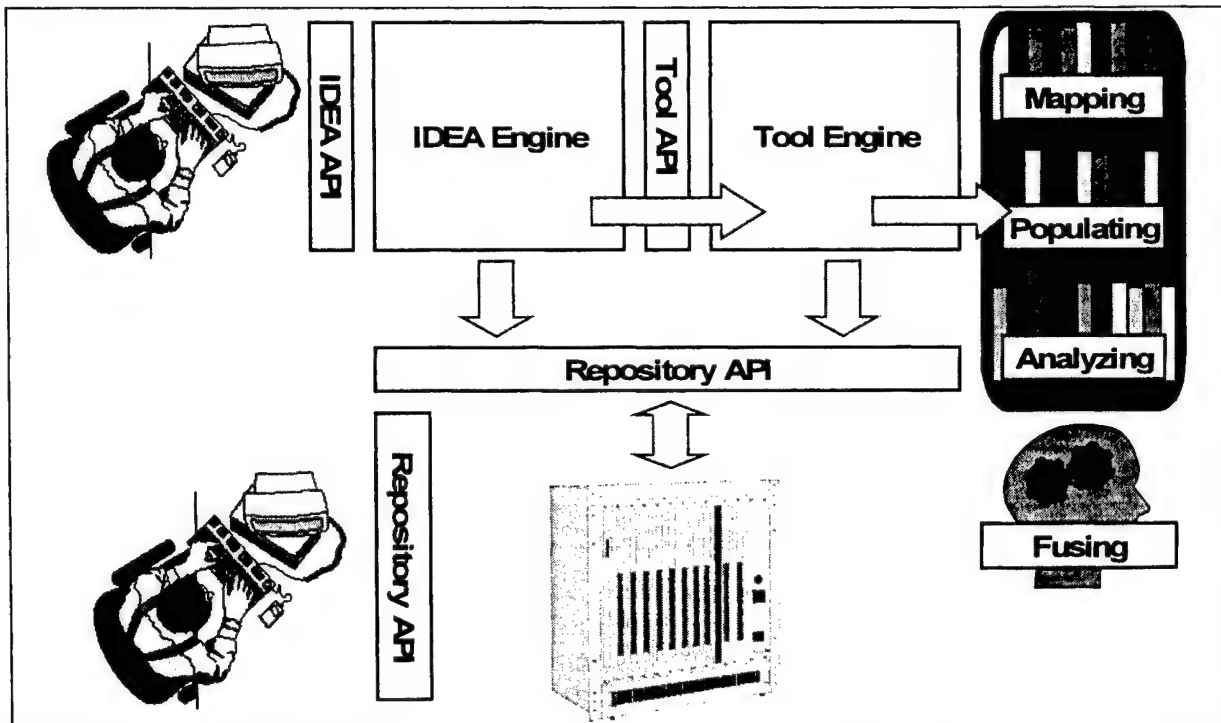


Figure 1.1-1 – Initial scope of IDEA as presented at the 2nd Blue Ribbon session

To understand the significance of the effort's change in emphasis, it is important to understand how IDEA was first intended to support security engineers.

IDEA was initially a continuation of a previous development known as Network Vulnerability Tool (NVT), with the addition of a Design focus. It was to aid security engineers in the assessment of networks prior to, during, and after deployment. The feature set for IDEA proposed a capability to support network design and vulnerability assessment using a common repository.

The principal enabling factor of the IDEA effort was its centralized repository of information. This repository stored network design information as well as the results from executing an external design tool(s). The Common System Model (CSM) format defined to store network data was an effort to correlate the various external tool's outputs and data formats. This common data format was used for communication between the IDEA repository and external tools.

FuzzyFusion™, as an external tool, relied upon IDEA's common repository for its data. The benefit of having FuzzyFusion™ designed and subsequently implemented as an external-to-IDEA tool allowed the IDEA component to focus on its correlation of data between tools. Data stored by IDEA that was required by FuzzyFusion™ would be formatted and sent to the FuzzyFusion™ component as if the Fuzzy Fusion™ component were just another integrated tool in the framework

As a component, FuzzyFusion™ played a significant though small role in the initial IDEA effort. Its capabilities were segregated from IDEA by the component interface. However, as part of the overall IDEA solution to an Analysis Design Environment, it was noted that there was some information within the Common System Model that a FuzzyFusion™ component would require. This evolved to a design where tools have the capability to interface to the

repository. Allowing tools to retrieve IDEA repository information put two data-driven burdens upon IDEA:

- 1) Ensure a common data-solution for all external tools, and
- 2) Provide a flexible interface capable of handling tool requests.

Understanding the initial scope of IDEA and how FuzzyFusion™ integrates with it is very useful in understanding the impact of the re-emphasis. IDEA was initially designed to:

- Support network design,
- Provide application support for multiple security tools,
- Contain a common database for storing multi-tool results,
- Be capable of managing multiple networks independently,
- Provide a meta-language to support the generation and management of security policies,
- Provide FuzzyFusion™ technology for assessing vulnerability analysis tool results.

During the first phase of this effort, the full IDEA application was designed. The Architecture document for the initial design of IDEA is included in Appendix C.

FuzzyFusion™ was an additional capability for IDEA. As an external tool, FuzzyFusion™ would maintain its own data (not share a database with IDEA) and interface with IDEA for necessary information. IDEA was designed to address the formatting of vulnerability tool results into the Vulnerability Taxonomy. The FuzzyFusion™ component would be capable of accepting taxonomy information and returning a measure of the vulnerability for each node within a network.

At the time of the change in emphasis, the details of the IDEA-FuzzyFusion™ component interface were still under construction. After the change in direction, it became necessary to better identify the interface and characterize the behavior of the FuzzyFusion™ component.

1.2 Initial FuzzyFusion™ Capability

Figure 1.2-1 shows the initial FuzzyFusion™ capability, which was to accept the results from data collection agents and correlate the results. The single network representation was to be implemented by the CSM repository. The multi-tool correlation realized by the internal Vulnerability Taxonomy provided a common frame of reference for the numerous data formats of external tools.

FuzzyFusion™ technology supports vulnerability assessment for nodes within the network using both the single network representation and the data correlation technologies within the IDEA repository.

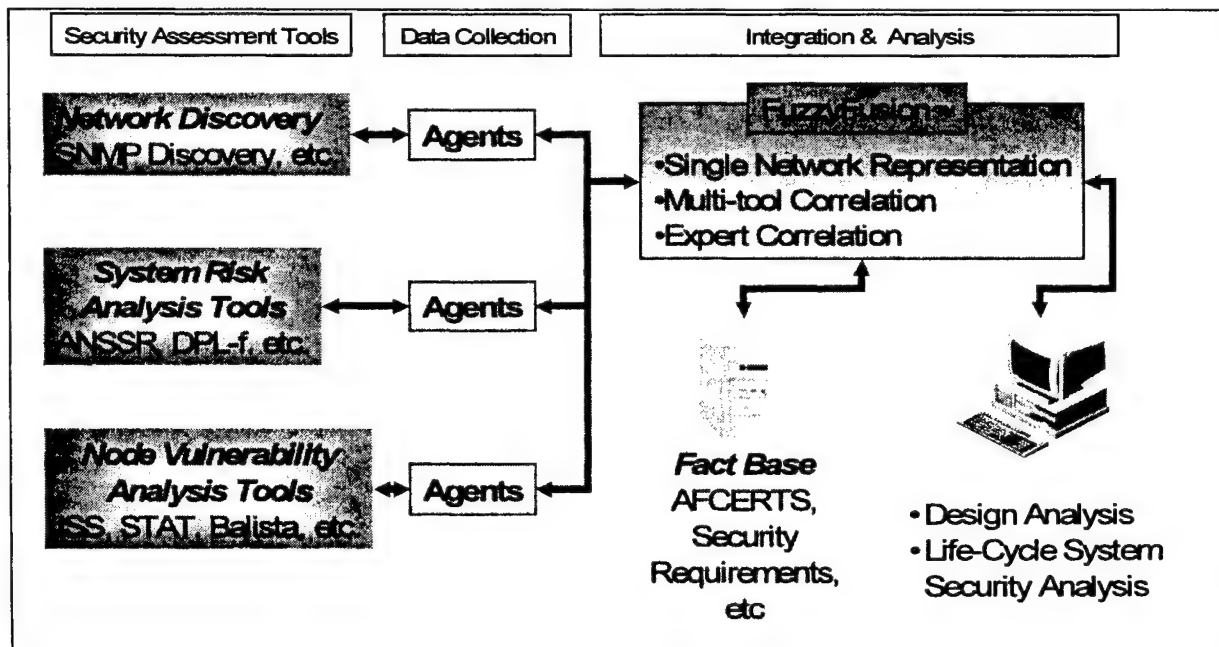


Figure 1.2-1 – Early view of FuzzyFusion™

IDEA supported a Blue Ribbon panel of experts who were selected to provide consultation. During its third meeting, with AFRL representation, it was decided to research additional concepts and mechanisms for an enhanced FuzzyFusion™ component. This component would be similar in function to that existing within Harris's commercial STAT® Analyzer product – a spin-off of the Network Vulnerability Tool effort. At the time, it was noted that STAT® Analyzer's FuzzyFusion™ component was still under construction and that the IDEA effort would complement that component with additional research at higher levels of abstraction in the data fusion model.

The FuzzyFusion™ black-box concept presented at the panel meeting (Figure 1.2-2) shows a naïve, yet possible, implementation of Fuzzy Fusion™ where the FuzzyFusion™ black box accepts data supporting the four defined levels of information data fusion.

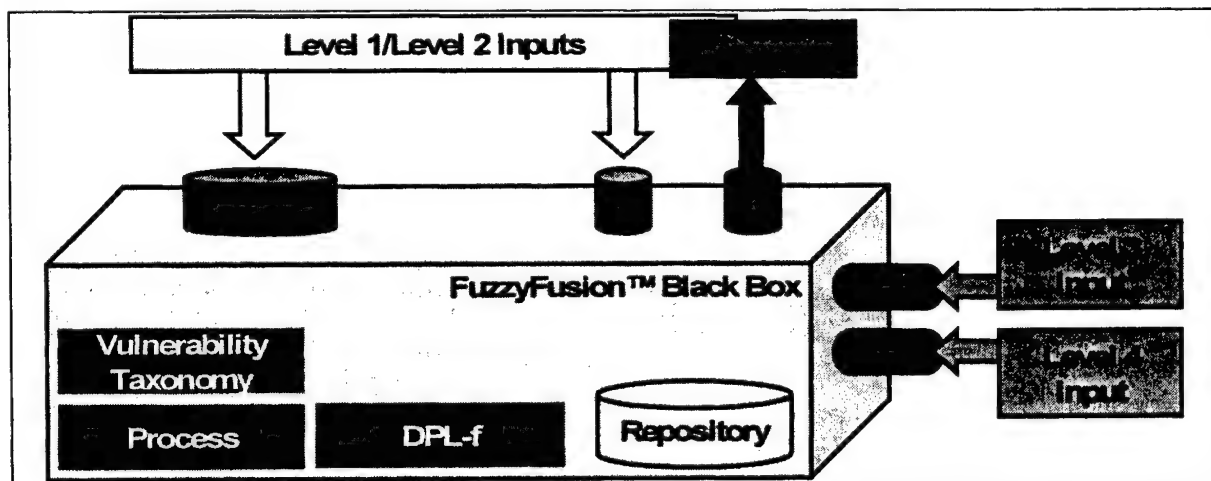


Figure 1.2-2 – The FuzzyFusion™ component supporting higher levels of data fusion

After IDEA's research direction was established we began exploring in more detail the various levels of data fusion (see Figure 1.2-3) and how they could be supported. To address this we brainstormed how the FuzzyFusion™ component would be used and how it could support each of the four levels of data fusion (see Figure 1.2-3).

- **Level 1 - Node Data Refinement**
 - Uses mapped taxonomies
- **Level 2 - Network Segment Refinement**
 - Adds an additional taxonomy that describes node (point) qualities
- **Level 3 - Risk Refinement**
 - Uses DPL-f and Threat Knowledge and Cost databases
- **Level 4 - System Refinement**
 - Uses DPL-f and Countermeasure, Component and Cost databases

Figure 1.2-3 – Data Fusion levels as presented in the 3rd Blue-Ribbon session

The result of our investigation into the levels of data fusion and how each level is supported by the FuzzyFusion™ component were the initial driving factors in our component development effort.

The final version of the FuzzyFusion™ component is guided by the research into data fusion levels as well as how the component supports potential use in a real-world application environment (as it is used in STAT® Analyzer or was to be used in the scope of the initial IDEA effort). This helped define strict boundaries between what data processing functionality the component contains and what data processing functionality users of the component are to contain. For example, STAT® Analyzer manages data filters and converters to convert vulnerability assessment tools into a common data format for use by FuzzyFusion™. IDEA's FuzzyFusion™ component is similar in that vulnerability mapping and data manipulation are performed external to the component. It is not the FuzzyFusion™ component's job to map tool vulnerability information into the internal Vulnerability Taxonomy. Rather, it is Fuzzy Fusion's™ job to correlate information into more comprehensible results.



Figure 1.2-4 – FuzzyFusion™ as a component technology

The re-emphasis of IDEA led to research into and development of a stand-alone FuzzyFusion™ component. The FuzzyFusion™ component is a tightly-focused module that accepts vulnerability information and produces a measure of vulnerability. Being a component, as opposed to a complete application, the FuzzyFusion™ component is more

flexible than a complete application in that its interface may be used in ways not enforced by a component. It is not possible for FuzzyFusion™ to constrain how it will be used by external, currently-non-existent applications. Also, as a component, it must rely on an external application (Figure 1.2-4) to prepare the vulnerability information. This document addresses these issues in further detail in Section 4.2. Both documentation for the Fuzzy Fusion™ and guidelines as to how the component is intended to be used are discussed.

1.3 FuzzyFusion™ Results

Early in this effort, it was unclear as to exactly what results were to be returned from the FuzzyFusion™ component. That the result of the FuzzyFusion™ process produces a measure of the vulnerability of the machine was obvious. However, it was initially unclear as to how to express that measure.

Two options were available to us at the time:

- 1) Return a single final-value representing the vulnerability measure of the node,
- 2) Return a list of final-values that may be used to calculate a final vulnerability measure, or further support an external vulnerability assessment tool.

We chose the second option as more viable because it would allow us to better analyze the results of FuzzyFusion™.

The FuzzyFusion™ component returns three types of information:

- A set of vulnerability categories with some related data. These are the intermediate results of the FuzzyFusion™ process.
- The result of fuzzy membership analysis across the vulnerability categories. These are the final results of the FuzzyFusion™ process. And
- A final value for the node derived from the fuzzy membership analysis. This value is calculated directly from the fuzzy membership analysis results.

Section 4.2 discusses the results produced by the FuzzyFusion™ component.

2 Document Overview

- Section 3 contains reference documents.
- Section 4 contains the program summary covering the levels of data fusion, what the FuzzyFusion™ component is and does, analysis of results, and contains the prototype information.
- Section 5 presents the conclusions of our research.
- Section 6 contains possible future work for FuzzyFusion™ technology.
- Appendix A contains database tables and database information used on FuzzyFusion™.
- Appendix B contains XML background and information.
- Appendix C is the original IDEA architecture document.

3 Reference Documents

Network Vulnerability Visualization Architecture (NVVA) Proposal – Volume I, Technical, Harris Corporation, Melbourne, FL, 1996.

Tallet, Joseph, Ronda Henning and Kevin Fox, “IDEA: An Information Superstructure”, Proceedings of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX II), June 2001.

Henning, Ronda, Margaret Knepper, and Kevin Fox, “Information Fusion Meta-Language Requirements”, IEEE Systems, Man, and Cybernetics Information Assurance Workshop, June 2001.

Vulnerability Analysis, Detection, and Reduction (VADR) Proposal – Volume I, Technical, Harris Corporation, Melbourne, FL, 1998.

4 Program Summary

4.1 Levels of Data Fusion

To date, traditional intelligence data fusion process modeling techniques have not been applied to the information assurance/operations disciplines. However, several calls for research have noted the problems of integrating disparate data sources into a common operational picture or situational assessment. To the best of our knowledge, interdisciplinary research between the data fusion community and the information assurance community has not been previously undertaken. The adaptation of traditional data fusion techniques to the information assurance correlation problem offers a potentially high payoff in the field of system vulnerability analysis.

The previous NVT effort developed the data fusion model presented in Table 4.1-1. The research resulted in a four-stage model of vulnerability fusion. This model is designed to reflect the global nature of a large enterprise, the increased dependence on inter-networks of information, and the vulnerabilities associated with individual network segments and devices. The model has been successfully applied to static vulnerability analysis, and has been integrated with current vulnerability assessment technologies for vulnerability identification and correlation.

Table 4.1-1 – The levels and purposes of the Intelligence Data Fusion process

DATA FUSION LEVEL		DESCRIPTION
1	Object Refinement	<ul style="list-style-type: none">• Transforms data into consistent frame of reference• Refines and extends, in time, estimates of object position, kinematics or attributes• Assigns data to objects to allow application of estimation process• Refines the estimation of object identity
2	Situation Refinement	<ul style="list-style-type: none">• Develops description of current relationships among objects and events in the context of the environment• A symbolic, reasoning process by which distributions of fixed and tracked entities and events and activities are associated with environmental and performance data in the context of an operational problem
3	Threat Refinement	<ul style="list-style-type: none">• Projects the current "situation" into the future and draws inferences about threats, vulnerabilities and opportunities for operations
4	Process Refinement	<ul style="list-style-type: none">• Monitors process performance to provide information for real-time control and long-term improvement• Identifies what information is needed to improve the multi-level fusion product• Determines the source specific data requirements to collect required information• Allocates and directs the sources to achieve mission goals

Level 1 - Object Refinement

The FuzzyFusion™ component assesses the risk for each node in the network. It is assumed that client applications of the FuzzyFusion™ component send vulnerability information relating to a single network node for further correlation. Used in this manner, the FuzzyFusion™ component returns a vulnerability assessment of a single node. This is the context in which IDEA's FuzzyFusion™ component was designed, namely, to facilitate

the development of single-node operational assessment information. The object refinement level of data fusion is demonstrated in the IDEA Fuzzy Fusion™ prototype.

Level 2 - Situation Refinement

For level 2 data fusion, some FuzzyFusion™ components within a network are configured to be dependent upon others. This configuration of component dependencies feature was beyond the scope of the IDEA FuzzyFusion™ component. However, its results could be fully conveyed through the return of a final vulnerability assessment: Data, User, or System. This result is forwarded to dependents of the FuzzyFusion™ component. Each dependent updates its own final vulnerability assessment in response to the new information.

After updating its final vulnerability assessment, the dependents will send their updated information on to their own dependents.

Level 3 - Threat Refinement

The FuzzyFusion™ component supports threat refinement by allowing its clients to provide an Opportunity For Exploitation¹ (OFE) with each vulnerability. The OFE is used to identify opportunistic exploitation paths through the network. The ability to predict the paths of nodes that an intruder will follow during an attack is a level 3 data fusion issue.

For level 3 data fusion, a centralized application (IA server) collects the FuzzyFusion™ vulnerability measures from each node. The IA server then identifies, based on the value of the OFEs, the possible intrusion paths through the network.

Another issue attributed to level 3 data fusion is the refinement of the exposures and their potential for harm within the network. Nodes containing sensitive information (company private, etc.) are ranked as goals for an attack. Were an attacker to gain access to the node, vital corporate information may be compromised. Once these nodes are identified, the IA server uses the information as it determines the exploitation paths through the network.

Level 4 – Process Refinement

Level 4 data fusion builds on the previous 3 levels of data fusion. After the assessment of exploitation paths through a network, a server application would examine the vulnerabilities for each node and generate a report specifying how to address the vulnerabilities. For level 4, causal information about the results of the FuzzyFusion™ analysis would be necessary. This kind of capability was discussed at the fourth Blue Ribbon panel meeting and a possible solution is provided in Section 6.3.

Data Fusion – An Example

The next section models a distributed network vulnerability application using the technology of FuzzyFusion™ in support of the four data fusion levels.

The remainder of this section discusses the four levels of data fusion and how the FuzzyFusion™ component supports each.

¹ The term OFE, a suggestion of Dr. Jim Bezdek, came about during the fourth Blue Ribbon Panel, during a discussion of terminology. Probability/Possibility of exploitation had domain implications that were not intended in this context.

4.2 Operational Concept for a FuzzyFusion™ Component

During the initial restructuring of the IDEA program, we investigated the various levels of data fusion and how they are supported by a FuzzyFusion™ component.

This section models a real-world application of the data fusion and FuzzyFusion™ technologies and shows how they are used to support real-time network vulnerability analysis.

4.2.1 Introduction

To increase the FuzzyFusion™ component's capacity for interfacing to other applications, we chose to use the eXtensible Markup Language (XML) for our interface. XML allowed us to define how the data would pass from the host application to our component (Figure 4.2-1).



Figure 4.2-1 – XML is the data transport mechanism for the FuzzyFusion™ Component

XML allowed us to reduce our efforts in passing the data to our FuzzyFusion™ component from the host application. For us, the host application was the testing application. In a real-world application, the host would be the application that collects vulnerability information and passes it to a FuzzyFusion™ component client.

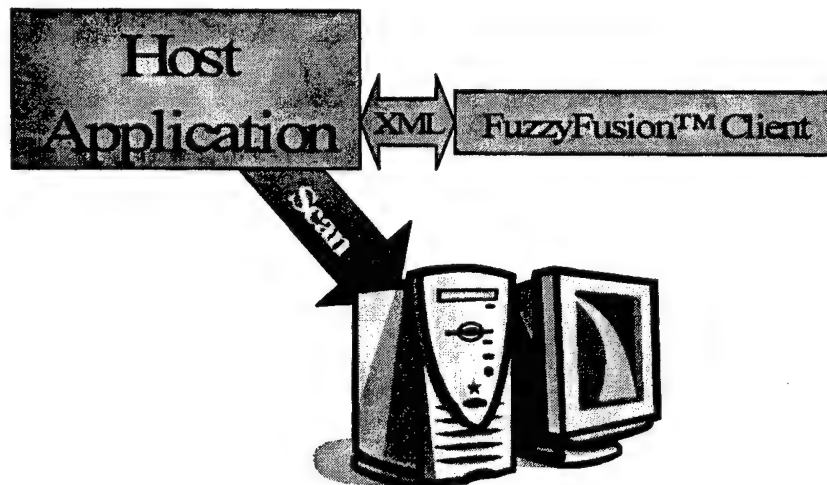


Figure 4.2-2 – The host application uses a vulnerability scanner to generate data for the FuzzyFusion™ Component

Another potential follow-on concept to XML is supporting vulnerability analysis in a distributed system. Each network node contains a vulnerability scanner and a FuzzyFusion™ client (Figure 4.2-2). As a client, the FuzzyFusion™ component is small, fast and readily adaptable to a distributed environment.

This section discusses how we view real-time vulnerability analysis of a network using FuzzyFusion™ component clients.

4.2.2 Distributed Real-Time Vulnerability Measurement

Consider a network of nodes where each node contains a small vulnerability scanner whose job is to periodically assess the vulnerability of its host. In the same manner that a virus scanner resides on individual networked machines, this lightweight-process also resides on each machine in a network. As vulnerabilities are detected, they are processed by the host application (acting as a Vulnerability Assessor) and a vulnerability measurement is assigned to the node. As this assessment changes, reflecting the node becoming more or less vulnerable, the information is sent to a network assessment tool. The network assessment tool notes the updated node information and re-assesses the vulnerability across the network.

Being a server-side application, the network assessment tool is capable of receiving and processing the vulnerability status of all nodes. The tool determines which nodes in the network are most vulnerable and reports its assessment to a network administrator. The network administrator analyzes the vulnerability information returned from the vulnerability scanning tools and selects vulnerabilities to address immediately.

4.2.3 Level 1 FuzzyFusion™ - What is the vulnerability of each node?

Although the FuzzyFusion™ component can support level 2 and 3 data fusion concepts, with inroads into level 4 data fusion, our research emphasized design goals and conceptually prototyped the level 1 component aspects. This allowed us to have early success with the fusion concepts, and provided a framework for further experimentation with higher levels of data fusion.

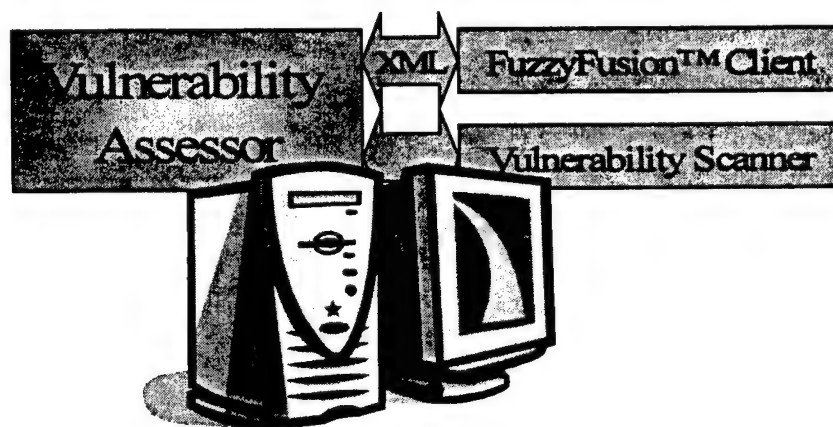


Figure 4.2-3 – Distributed Real-Time Vulnerability Assessment Application

The level 1 FuzzyFusion™ component developed on IDEA is a small client-side component that processes a node's vulnerability information and generates a measurement of vulnerability. The measurement of vulnerability is called the Fuzzy Vulnerability Indicator (FVI).

The following section discusses the importance of a level 1 FuzzyFusion™ component in network vulnerability assessment.

4.2.3.1 Level 1 Scenario

Suppose each node in a computer network contains an application continually scanning the host for vulnerabilities. This application, the Vulnerability Assessor shown in Figure 4.2-3, formats the vulnerabilities returned from the Vulnerability Scanner and forwards them to its FuzzyFusion™ client.

After the FuzzyFusion™ client performs an analysis of the node's vulnerabilities it generates an FVI and returns it to the Vulnerability Assessor. It is the task of the Vulnerability Assessor to decide how to address the FVI value. If the FVI value is the same as previous values, or after applying some tolerance calculations is found to be the same, the Vulnerability Assessor may do nothing.

If the FVI value is significantly different from previous FVIs the Vulnerability Assessor may be programmed to send the new information to another application or a network administrator.

This is the scope of level 1 data fusion analysis: Generate an FVI for a node and notify a network administrator about it when it exceeds a predefined threshold value.

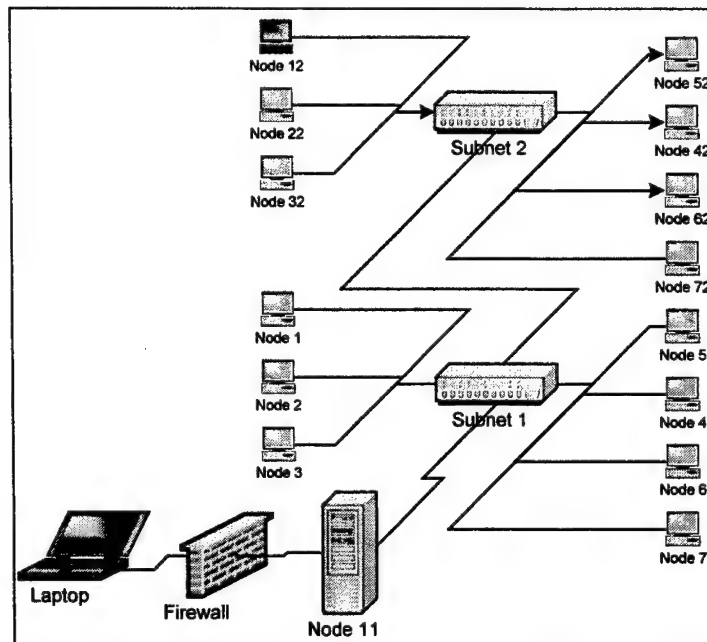


Figure 4.2-4 – Network Configuration

Figure 4.2-4 depicts one possible view a network administrator may create as a result of notification from a Vulnerability Assessor. In the figure, Node 12 is found to have a vulnerability distinguishing it from the other nodes. For level 1 data fusion it is the responsibility of the network administrator to analyze and respond to the information.

4.2.4 Level 2 FuzzyFusion™ - How do vulnerabilities in one node affect other nodes?

To address level 2 FuzzyFusion™, the concept of node-proximity is introduced. As discussed in the Blue Ribbon panel, there are several methods of determining *nearness*

across nodes. One method takes into account the topology of the network. Another takes into account the exploitations or strengths available on neighboring nodes.

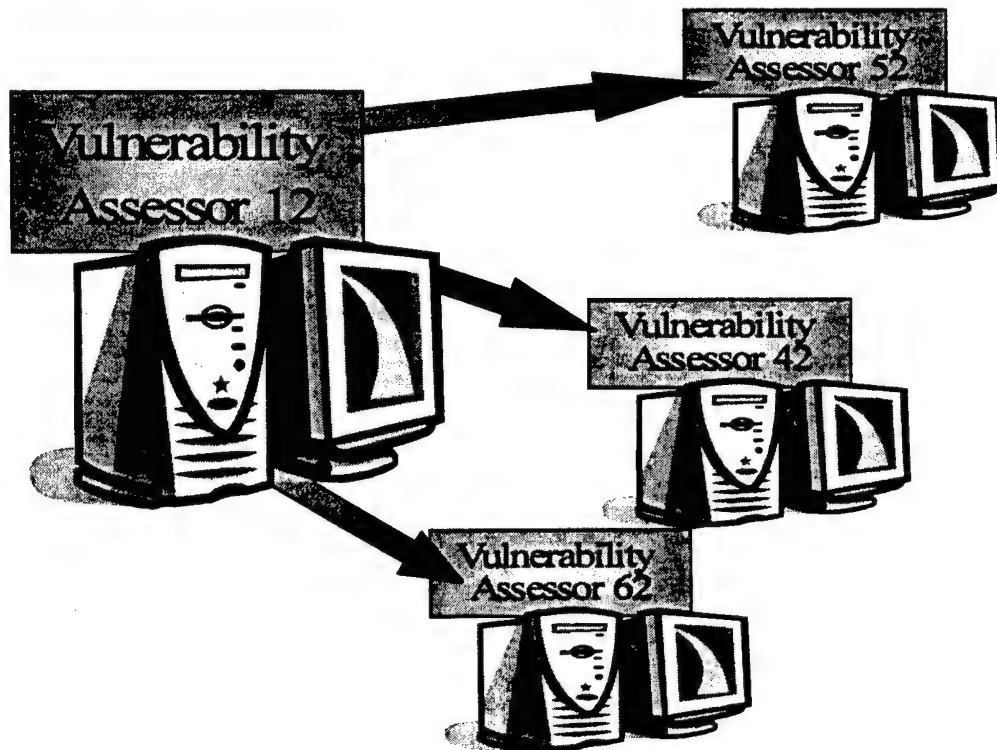


Figure 4.2-5 – Vulnerability Assessors linked together

Level 2 FuzzyFusion™ allows the Vulnerability Assessor to be configured as dependent on other Vulnerability Assessors.

4.2.4.1 Scenario

In Figure 4.2-5, Vulnerability Assessors on nodes 42, 52, and 62 are dependent on Vulnerability Assessor node 12. This means that, in this network of four nodes, three nodes depend on one node to remain secure. For an intruder to gain access to nodes 42, 52, or 62, they would have to exploit a vulnerability within node 11. This network topology is a simple client-server architecture. The client nodes (42, 52, and 62) are accessible through the server (12).

Given this topology, a network administrator would configure the Vulnerability Assessor on node 12 to forward its assessment value to nodes 42, 52, and 62. If node 12 became vulnerable, it would increase the level of exposure to nodes 42, 52, and 62.

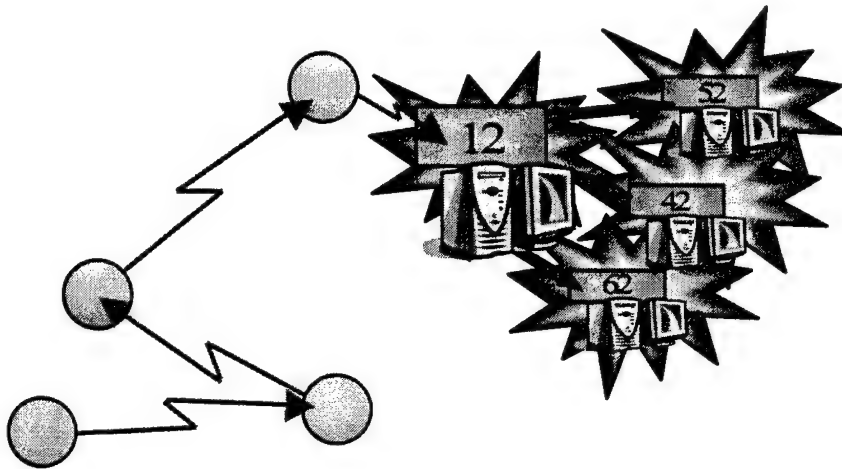


Figure 4.2-6 – Exploitation path through a network

Figure 4.2-6 shows a possible vulnerability path through a network. If an intruder gains access to node 12, nodes 42, 52, and 62 become potential targets for attack.

In this scenario, the Vulnerability Assessor on node 1 would forward changes of its FVI to the Vulnerability Assessors on the three dependent nodes. Each dependent node would then re-assess their FVI possibly raising flags to the network administrator.

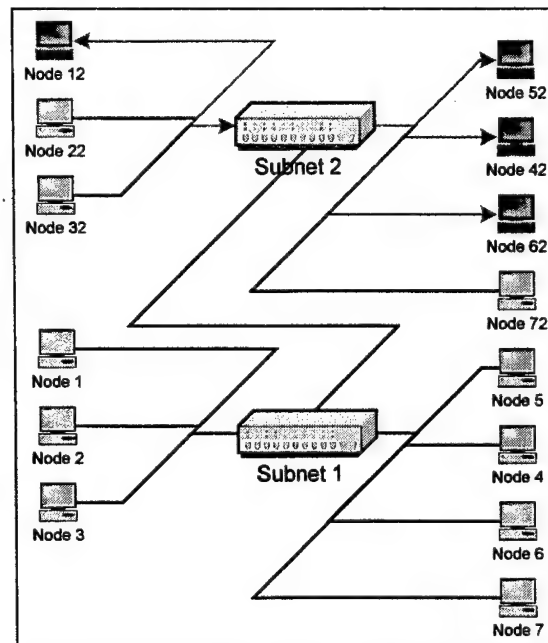


Figure 4.2-7 – Dependent node, Node 52 raises its vulnerability level

Figure 4.2-7 depicts one possible view a network administrator may have as a result of the increase in exposure to Node 12. The increase caused an increase in Node 52. Nodes 42 and 62, although dependent upon Node 12, are not impacted by the change. The network administrator would ascertain the problem with Node 12 and fix it. After fixing Node 12, the Vulnerability Assessor on Node 52 would lower its exposure value.

Compare this with level 1 FuzzyFusion™. In Figure 4.2-4 the information to the network administrator would be simply that Node 12 has become significantly vulnerable to attack. The importance of that vulnerability, determined by dependents of the node, is not provided. The network administrator would not realize that when Node 12 became vulnerable to attack, nodes 42, 52, and 62 also became vulnerable. Level 2 FuzzyFusion™ information is useful in assessing the extent of the vulnerability across the network's nodes.

This scenario shows how a network of dependent nodes may be managed using level 2 FuzzyFusion™ technology. As exposures are identified, nodes dependent on the strengths of other nodes for protection from external intruders would automatically re-assess their level of exposure.

4.2.5 Level 3 FuzzyFusion™ - How do I get to a vulnerable node that allows me access to critical data?

Level 3 FuzzyFusion™ incorporates the technology of level 2 FuzzyFusion™ by defining a single collection place for all level 1 and level 2 values. It also adds the capability of configuring a node's value to the company. The use of this value (the dollar amount lost by the company if the node is compromised) is discussed later in this section.

Level 2 FuzzyFusion™ identifies paths from a vulnerable node to neighboring computers, revealing a weak point that, if exploited, allows an intruder access to other machines. Level 3 performs the same function with the additional capability of pointing out nodes of maximum impact to companies if successfully attacked.

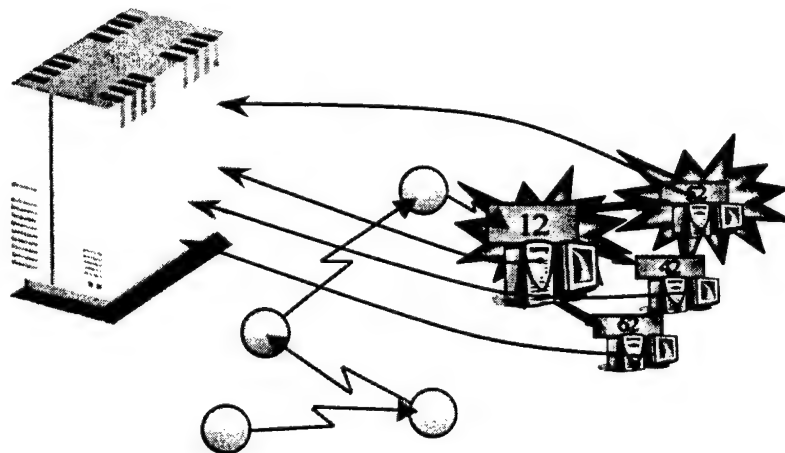


Figure 4.2-8 – Key node on network becomes more vulnerable

Within our network shown in Figure 4.2-8, the Vulnerability Assessor on node 12 has identified an exposure and updated its FVI. After sending the new FVI to the dependent nodes (42, 52, and 62) the Vulnerability Assessor on node 52, after re-evaluating its FVI has raised its level of exposure.

For level 3 FuzzyFusion™, each Vulnerability Assessor sends its information to a server-application; a Network Assessor application. The Network Assessor collects FVI results from each node in the network. The capabilities of the Network Assessor would be to build a map of the network showing vulnerable nodes as well as nodes containing key corporate assets.

Moving from level 2 to level 3 FuzzyFusion™ is more difficult than moving from level 1 to level 2. The Vulnerability Assessor would have a minor change – send status to server. In essence, the Network Assessor would become a dependent of the Vulnerability Assessor. The Network Assessor is not a simple component. It is capable of capturing potentially large amounts of FVI information, assessing, and rendering the information for a network administrator.

The Network Assessor would have to automatically identify paths of exploitation through the network. This information is strongly similar to that discussed during the third Blue Ribbon Panel where Dr. Blaine Burnham discussed the topic of node proximity.

Level 3 FuzzyFusion™ must determine proximity relationships between nodes. There are two promising approaches to generating this information: fault tree analysis and cognitive maps. Fault tree analysis is a commonly used analysis method for determining combinations of various methods of hardware and software failure. Cognitive maps are a useful means of analyzing causal knowledge.

Using Fault-Tree Analysis or Cognitive Maps, the paths are identified and other information may be divined:

- Types of exploitations necessary to overcome to reach each vulnerable node,
- Effort to overcome the vulnerabilities across the path – use proximity information,
- Probable attack patterns - Include Opportunities For Exploitation information,
- Path Variations – Analyze different approaches to the same goal,
- Selects Optimal Attack Path – Identify the most opportunistic path to the goal.

This network analysis need not limit itself to simply identifying network-external candidate attacks. Internal-network attacks are just as, if not more, important for vulnerable paths. It is not sufficient to say that internal paths to a vulnerable system are a subset of external paths. Internal paths may start at locations the external path may not reach. However, there is probably a set of nodes – or a single subnet – that can be identified for which all others are called external. Then, analysis breaks down into an identification of the path through the subnet plus the local-external path.

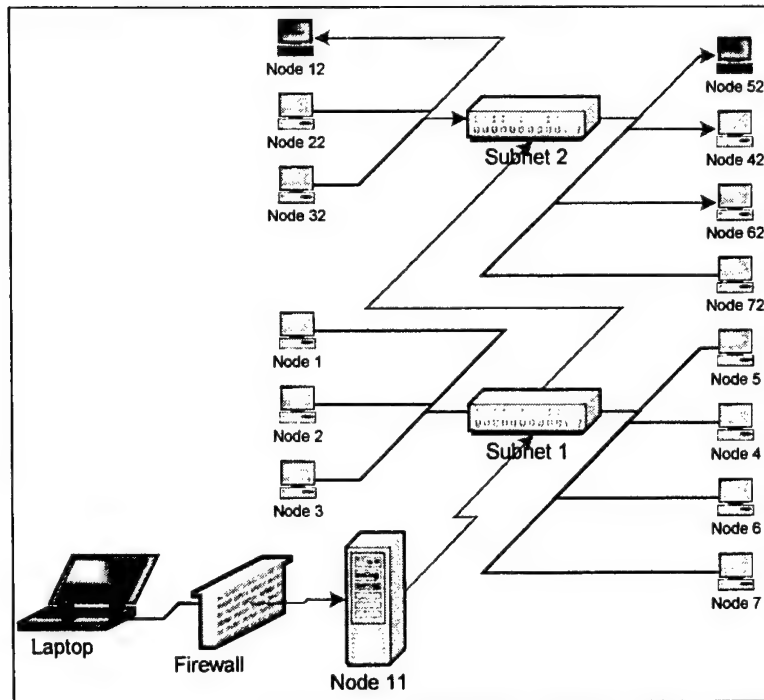


Figure 4.2-9 – External to Internal attack path to a key node

Within our network, Node 12 has visibility into three nodes (42, 52, 62) one of which, Node 52, contains critical information. Although Node 52 itself may have no easily exploitable vulnerabilities it may become more prone to attack once vulnerabilities on Node 12 are exploited. The network view (Figure 4.2-9) generated by the Network Assessor application shows the network administrator an opportunistic path to Node 52.

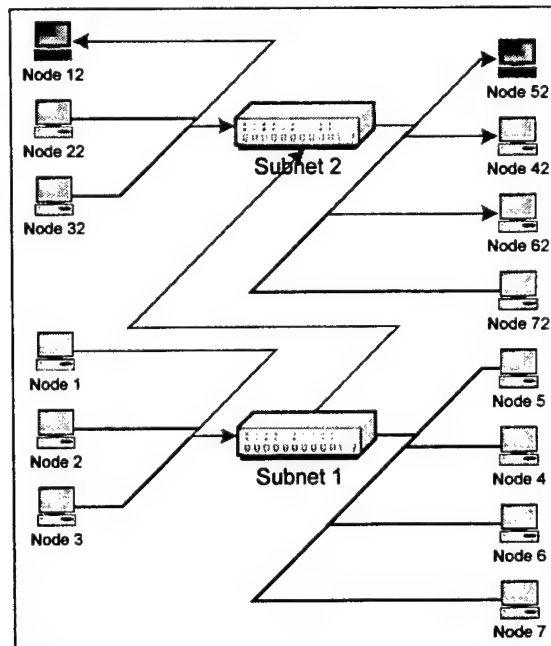


Figure 4.2-10 – Internal to Internal attack path to key node

Figure 4.2-10 shows a possible representation for an internal attack to the vulnerable node.

With this information, the network administrator is able to quickly identify and fix exploits on key network nodes.

4.2.6 Level 4 FuzzyFusion™ - How do I increase the level of difficulty of following a path to the critical node?

Level 4 of FuzzyFusion™ continues where Level 3 FuzzyFusion™ stopped. Level 3 leaves the decision to the network administrator for selection of optimal vulnerability fixes. Level 4 FuzzyFusion™ adds the capability to the FuzzyFusion™ component to be able to determine which vulnerabilities led to the FVI and return them to the Vulnerability Assessor. The Vulnerability Assessor would, in turn, send them to the Network Assessor for collection and processing.

The Network Assessor, after processing the information, would direct the Vulnerability Assessors to, where possible, automatically fix the vulnerabilities. Support for this feature would be required of COTS vulnerability assessment tools. For example, STAT® Scanner has an auto-fix capability for some vulnerabilities. For vulnerabilities without auto-fixes, the Network Assessor would kick off a workflow to have them manually fixed.

4.2.7 Summary

The level 1 FuzzyFusion™ component prototyped on IDEA is a small component in a larger network vulnerability application. Its support for level 1 FuzzyFusion™ is the conceptual starting point for a complete real-time vulnerability assessment application. The impacts to the FuzzyFusion™ component when moving up in levels of data fusion abstraction are minimal. Most of the impacts associated with each level are external to the FuzzyFusion™ component.

Level 2 requires the FuzzyFusion™ component to use an externally provided starting value. This interface, although not implemented, is provided by the component.

Level 4 requires the FuzzyFusion™ component to generate information pertaining to how the FVI was impacted by vulnerabilities. Details for this simple modification are discussed in Section 6.3.

4.3 Technical Approach

The approach undertaken for this effort is to provide a measure of vulnerability for a single node. This measure is the result of analyzing the vulnerability, level of trust or confidence in the vulnerability (based on a security engineer's assessment of the tool's capabilities), and Opportunity For Exploitation.

For this effort, this measure is a multi-faceted value containing FuzzyFusion™'s intermediate conditions as well as an overall assessment of the node's vulnerability.

Our objective is to apply FuzzyFusion™ to determine the vulnerability of a single node. The vulnerability data for the FuzzyFusion™ component is collected externally by analysis tools. The FuzzyFusion™ component accepts this data in the IDEA Vulnerabilities and Exposures (IVE) format. This data format encompasses two distinct data formats: the Common Vulnerabilities and Exposures (CVE) format and an IDEA-specific data format termed the IDEA Analysis Taxonomy (IAT) (see Figure 4.3-1).

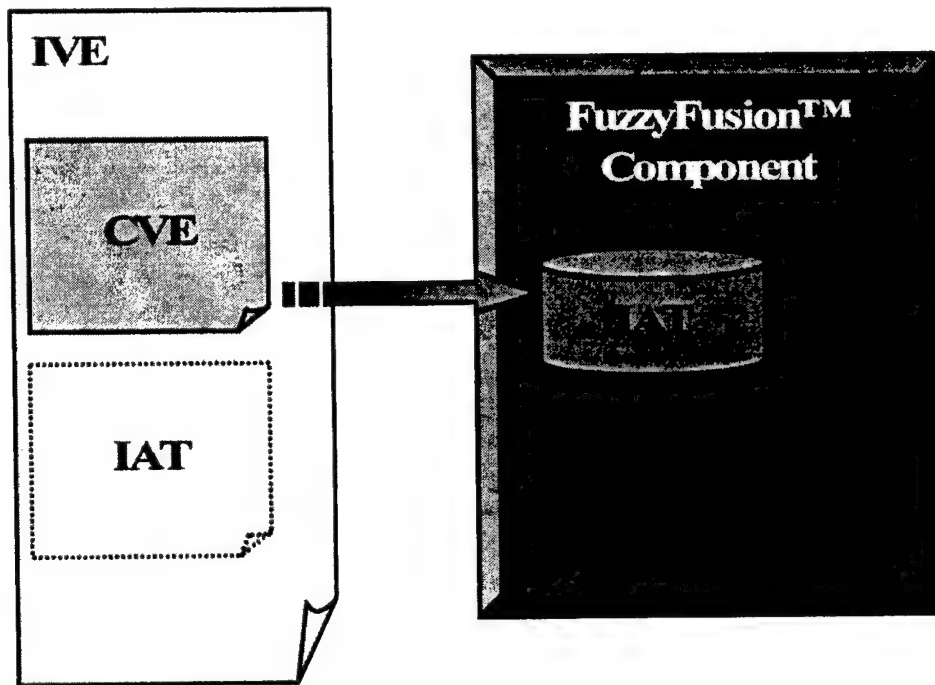


Figure 4.3-1 – Data for the FuzzyFusion™ Component

Information about CVE may be found on the <http://cve.mitre.org/> website. The IVE format is a simple internal data format used to reason about external vulnerability scanner results.

This section describes our approach to accepting vulnerabilities and related information and producing a measure of the node's overall vulnerability.

4.3.1 The FuzzyFusion™ component database

The FuzzyFusion™ component accepts vulnerability information in either CVE or IVE format. Internally, both CVE and IVE information is mapped to the IAT format.

Table 7.1-1 shows the enumeration for the IVE's and their descriptions. CVEs are mapped to these by the users of the FuzzyFusion™ component. Users may also pass vulnerabilities using the IVE identifier, also shown in the table.

Within the FuzzyFusion™ component database there is stored a repository of *a-priori* knowledge. This knowledge describes the state changes which the FuzzyFusion™ engine undergoes during its calculations. The data is an accumulation of knowledge from the NVT, STAT® Analyzer and STAT® Scanner programs.

Section 7.3 shows the rulebase for the FuzzyFusion™ engine. The format of the table is shown below.

Table 4.3-1 – IAT Mapped to Pre and Post Conditions

IAT-ID	PreCondition	PostCondition
--------	--------------	---------------

The IAT-ID – IDEA Analysis Taxonomy (IAT) are the building blocks of the vulnerability analysis. Vulnerability analysis tools identify vulnerability information utilizing system configuration and system status information. Databases of known vulnerabilities are referenced and a vulnerability assessment is produced. These vulnerability assessments are lists of specific system vulnerabilities. It is these vulnerabilities that are translated into IVE information by FuzzyFusion™ component users prior to sending them to the component.

The PreCondition is used as a starting point for the FuzzyFusion™ analysis. The PostCondition is an intermediate condition for the FuzzyFusion™ analysis.

IDEA uses a simple pre-condition of *Start* and a more complex post-condition. The post-conditions are categories of vulnerabilities that each IAT is mapped to.

There are 16 categories of vulnerabilities for FuzzyFusion™. They are listed in Table 6.2-1.

4.3.2 Client vulnerability mapping guidelines

Each tool vulnerability contains a one or more qualities which we call IATs. IATs are types of vulnerability. As such they may be combined to form the characteristics of a single tool vulnerability. Generally, however, tool vulnerabilities are mapped to the single, most destructive exploitation involved. For example, vulnerabilities which lead to administrative or root access to a node rank higher than vulnerabilities leading to user-level access. Vulnerabilities leading to user-level node access are ranked higher than vulnerabilities leading to access to network traffic.

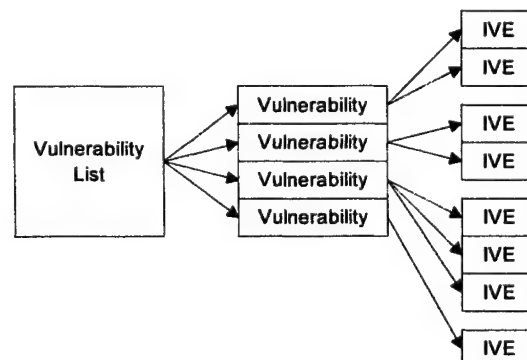


Figure 4.3-2 – Mapping tool vulnerabilities to IVEs

Vulnerability tools produce lists of vulnerabilities for a give node. Each vulnerability must be mapped to an IVE. The following sections discuss, in greater detail, this vulnerability mapping process.

4.3.2.1 Example of mapping a tool-generated vulnerability to an IAT

STAT® Scanner performs an analysis of a computer on a network: known as a node. It identifies a set of vulnerabilities for it based upon an internal database. This set of vulnerabilities has been mapped, by the STAT® Analyzer team to a vulnerability taxonomy. This example shows how a STAT® Scanner vulnerability is mapped to the FuzzyFusion™ component's IVE.

STAT® Scanner's vulnerability number 885 in Table 4.3-2 describes one NT vulnerability leading to denial of service.

Table 4.3-2 – STAT® Scanner vulnerability description

Vuln ID	Description
885	An administrative script in Internet Information Server (IIS) does not correctly handle a missing argument. This could cause the script to go into an infinite loop, consuming all CPU resources on the server. The permissions on some of the IIS tools could allow web site visitors to use these tools to cause a denial of service. For more information see Microsoft Security Bulletin MS00-044 and Knowledge Base Article Q267559.

This vulnerability may result in a Denial of Service for the node and is therefore mapped to IVE-270.

Table 4.3-3 – IVE and description

IVE ID	Description
IVE-270	Protect against Denial of Service

Some vulnerabilities may contain more than one vulnerability characteristic. CVE-1999-0898 can allow a buffer overflow which may be exploited for privilege elevation. In most instances a single CVE is mapped to the vulnerability with the greatest potential for harm. This is a guideline used in all but a couple of cases on IDEA.

4.3.2.2 Example mapping a CVE vulnerability to an IAT

As mentioned before, our guideline is to map tool-generated vulnerabilities to the most destructive vulnerability type, i.e. Administrator access or Privilege Elevation. CVEs, however, may be mapped to multiple vulnerability types if the CVE is sufficiently complex enough to warrant a multiple mapping.

For this effort we almost always avoided mapping CVEs to multiple IVEs unless the vulnerability tool mapped multiple vulnerabilities to a single CVE. For these cases, we mapped the tool identified CVEs to multiple IVEs using the tool's vulnerability database.

For example, STAT® Scanner's vulnerability database maps CVE-1999-898 to two individual vulnerabilities. We, in turn, mapped the CVE to two IVEs.

Table 4.3-4 – STAT® Scanner vulnerability information

STAT® Scanner's Vulnerability ID	CVE ID
684	CVE-1999-0898
685	CVE-1999-0898

This project makes use of the STAT® Analyzer database information for mapping tool vulnerabilities to IVEs. STAT® Analyzer maps STAT® Scanner's vulnerabilities, 684 and 685 to a vulnerability taxonomy which we have numbered. IVE-295 and IVE-130 are the corresponding numbers. Table 4.3-5 shows the description for the CVE and Table 4.3-6 shows the description for the two IVEs that are mapped to the CVE.

Table 4.3-5 – CVE Description

CVE ID	Description
CVE-1999-0898	Buffer overflows in Windows NT 4.0 print spooler allow remote attackers to gain privileges or cause a denial of service via a malformed spooler request.

Table 4.3-6 – IVE Description

IVE ID	Description
IVE-295	No Buffer Overflow
IVE-130	No privilege elevation without I+A

Mapping tool vulnerabilities to FuzzyFusion™ IVEs is not performed within the FuzzyFusion™ component. Users of the component must map vulnerability information prior to calling FuzzyFusion™. The Vulnerability Assessor shown back in Figure 4.2-3 is an application that would perform the vulnerability scanner to FuzzyFusion™ component mapping. STAT® Analyzer is an example of a tool that performs this mapping.

Table 4.3-7 – Mapping Summary

CVE ID	STAT® Scanner Vulnerability ID	IVE ID
CVE-1999-0898	684	IVE-295
	685	IVE-130
None Associated	885	IVE-270

Table 4.3-7 shows the tool vulnerability to IVE mapping. All of STAT® Scanner's vulnerabilities are mapped to IVEs. For those STAT® Scanner vulnerabilities that have a corresponding CVE, the CVE is mapped to the IVEs also. Except for the CVE-to-IVE mapping, the STAT® Scanner vulnerability to IVE mapping is performed by the STAT® Analyzer team.

Users of the FuzzyFusion™ component will use a similar mapping process in order to send the FuzzyFusion™ component vulnerability information.

Table 7.1-1 contains the list of IVE values available to users of the FuzzyFusion™ component.

4.3.3 The FuzzyFusion™ component

The FuzzyFusion™ component exports a well-defined interface that supports the following capabilities:

- ✓ Storage of vulnerability information
- ✓ Relating each vulnerability to a tool
- ✓ Relating Opportunities For Exploitation (OFE) information to a vulnerability
- ✓ Relating Trust to a vulnerability
- ✓ Generation of results for a single or set of tools

The component accepts its data in XML format (see Appendix D) and returns results in XML format. XML was chosen for its data-independence.

In addition to vulnerability information, clients may pass in three vulnerability attributes.

- 1) Tool identifier for the tool generating the vulnerability – this is a numeric Vulnerability Assessor defined identifier that is used by FuzzyFusion™ to group the vulnerability information.
- 2) Trust in the vulnerability – this is a human-generated or Vulnerability Assessor-generated assessment of the chance that the vulnerability actually exists (Range is 0..1).
- 3) Opportunity For Exploitation (OFE) – this is a human-generated or Vulnerability Assessor-generated value depicting the chance that the nature of the vulnerability is such that it will be exploited (Range is 0..1).

These three attributes, along with the mapped vulnerability are used by the FuzzyFusion™ component to generate a vulnerability assessment for the node.

The vulnerability assessment contains the following information.

- The set of intermediate conditions along with their Grade and OFE – shows the vulnerability categories, the approximate number of vulnerability hits in each category, and the maximum OFE across the vulnerabilities leading to the category.
- The set of end conditions along with their FuzzyFusion™ value and OFE – (Data_End, User_End, System_End),
- A final concluding value for the node, and
- The maximum end condition for the node – to be used for Level 2 FuzzyFusion™

Table 4.3-8 – Sample FuzzyFusion™ component output (after translated from XML)

Account	16	1
Administrator	2	1
Audit	14	1
Backdoor	0	0
Compromised	0	0
Data	6	1

Encryption	2	1
File	12	1
Hardware	4	1
Hijack	37	1
Network	3	1
Password	12	1
Privilege	0	0
Process	18	1
System	13	1
TBI	1	1
Data_End	4.32	1
User_End	32.07	1
System_End	52.39	1
END	37.605	0
System		

The gray boxes in Table 4.3-8 are not used by the FuzzyFusion™ component.

The values of Data_End, User_End, and System_End are comparative values showing the chance that an attacker may exploit vulnerabilities leading to those results.

Data_End implies that the attacker will only be able to exploit vulnerabilities allowing them access to network traffic.

User_End implies that the attacker will be able to exploit vulnerabilities allowing them access to user accounts or user account information.

System_End implies that the attacker will be able to exploit vulnerabilities allowing them access to administrative access to the node.

The End conclusion is a representation of the strength of Data_End, User_End, and System_End.

The final value of the table is the FVI for the node. This is the FVI discussed in Section 4.2.4 detailing level 2 FuzzyFusion™. Section 4.3.5 goes into detail on the FuzzyFusion™ algorithm.

The next section describes the XML format of the inputs and outputs to the FuzzyFusion™ component.

4.3.4 Data input requirements for the component

The FuzzyFusion™ component accepts XML-compatible data. The tag hierarchy for the XML data is shown in Table 4.3-9.

Table 4.3-9 – Tag Hierarchy for IDEA's XML data

IDEAINPUT
IMPORT
FILENAME
EXPORT
FILENAME
STARTCONDITION
CONDITION
TOOLLIST
TOOLID
POLICY
IAT
VULNERABILITY
IVE
TRUST
OFE
TOOL
IDEARESULTS
RESULT
CONDITION
GRADE
OFE
ENDCONDITION

The XML tag IDEAINPUT is used to denote input for idea. Sub-tags are used to provide additional data.

This section describes the format and use of the XML inputs and results of the FuzzyFusion™ component.

IMPORT – this tag provides the name of the file containing XML information for the component.

EXPORT – this tag provides the name of the file for the component's result information.

STARTCONDITION - this tag supports level 2 FuzzyFusion™. If the local node is dependent on a remote node, the end condition of the remote node is used as the start condition of the component. STARTCONDITION is the mechanism by which the component's start condition is externally modified. Possible values for STARTCONDITION are: Data, User, or System.

TOOLLIST – this tag is used to direct the component as to which tool's vulnerabilities to run the FuzzyFusion™ process on.

POLICY – this tag is used to provide policy information to the component. Policy information is provided to the component in IAT-compatible format.

VULNERABILITY – this tag is used to provide vulnerability information to the component. Each vulnerability is identified by its IVE attribute. TRUST is used to document the level of trust the client has with the existence of the vulnerability. OFE is the possibility of exploitation for the vulnerability and TOOL is a numeric alias for the name of the resource that discovered the vulnerability.

The FuzzyFusion™ component accepts these inputs and generates a vulnerability measurement. The results are returned in an XML file with the following tags.

IDEARESULTS – this tag denotes the output from FuzzyFusion™.

RESULTS – This tag is used for the condition-based results list from the component. CONDITION names the resulting condition for which the values of GRADE and OFE apply.

ENDCONDITION – This tag is used to provide the final measure, or end condition of the FuzzyFusion™ operation. For level 2 FuzzyFusion™ the end condition is sent to dependent nodes by the client.

4.3.5 Processing the data

This section discusses the FuzzyFusion™ algorithm implemented in IDEA. We review the following:

- Definitions
- Inputs and outputs
- Processing overview
- Algorithm details
- The divergence of IDEA FuzzyFusion™ from STAT® Analyzer FuzzyFusion™.

4.3.5.1 Definitions

We define the following terms, which we use in the following discussion of the algorithm.

Policy list is a list of IVEs with which we are concerned.

Tool List is a list of Tools producing Hits with which we are concerned.

Hit is an IVE found to exist by a specific Tool. A Hit has an associated Tool, Trust and Possibility.

Pruned Hit List is a list of those IVEs included in the current Policy and whose associated tool is on the Tool list.

$\text{Pruned Hit List} = (\{\text{Hits}\} \cap_{\text{IVE}} \{\text{Policy}\}) \cap_{\text{TOOL}} \{\text{Tool List}\}$
--

Figure 4.3-3 – Pruning formula

Intermediate Condition is one of 16 Conditions shown in Table 7.2-1.

Rule is a function which maps one IVE to one Intermediate Condition.

$\text{Intermediate Condition}_k = \text{Rule}_i(\text{IVE}_j)$

Figure 4.3-4 – Intermediate Condition Formula

In the current implementation, there can be up to 16 rules for each IVE.

Relevant Rule List is a list of rules where the IVE to which the rule is applied is an element of the Pruned Hit list.

Reachable Intermediate Condition is one of the Intermediate Conditions, when a Relevant Rule exists which produces that Condition.

4.3.5.2 IDEA FuzzyFusion™ processing inputs

FuzzyFusion™ requires a Pruned Hit List and a Relevant Rule List as inputs for processing. In the current implementation, the FuzzyFusion API is responsible for supplying those inputs to the FuzzyFusion engine. As discussed below, the STAT® Analyzer FuzzyFusion™ required a start condition, but that artifact was not used in the IDEA FuzzyFusion™ algorithm.

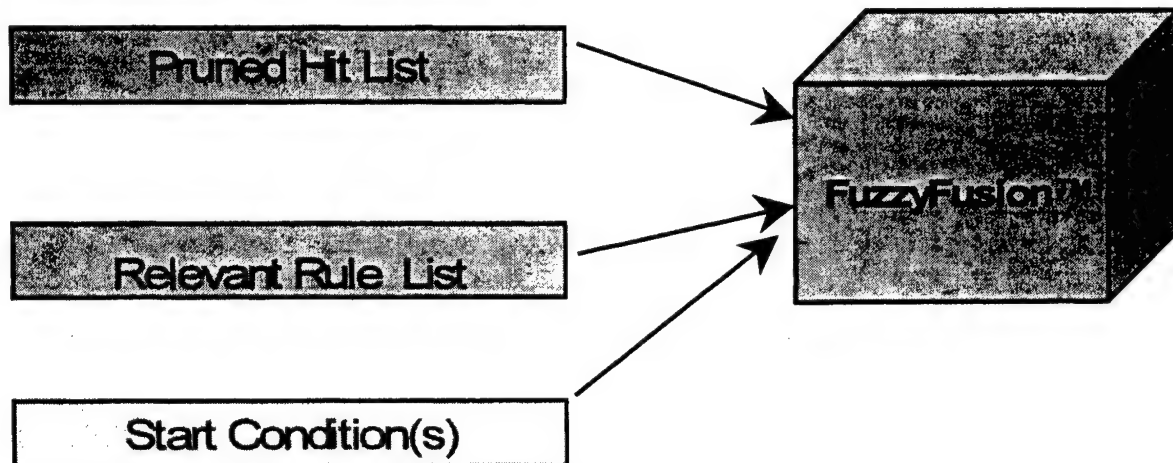


Figure 4.3-5 – Diagram of FuzzyFusion™ inputs.

4.3.5.3 IDEA FuzzyFusion™ processing outputs

FuzzyFusion™ produces three types of result set outputs. One set provides information for each of the Intermediate Conditions; another set provides information for three summary conditions, and the last set provides a single summarizing value for the node.

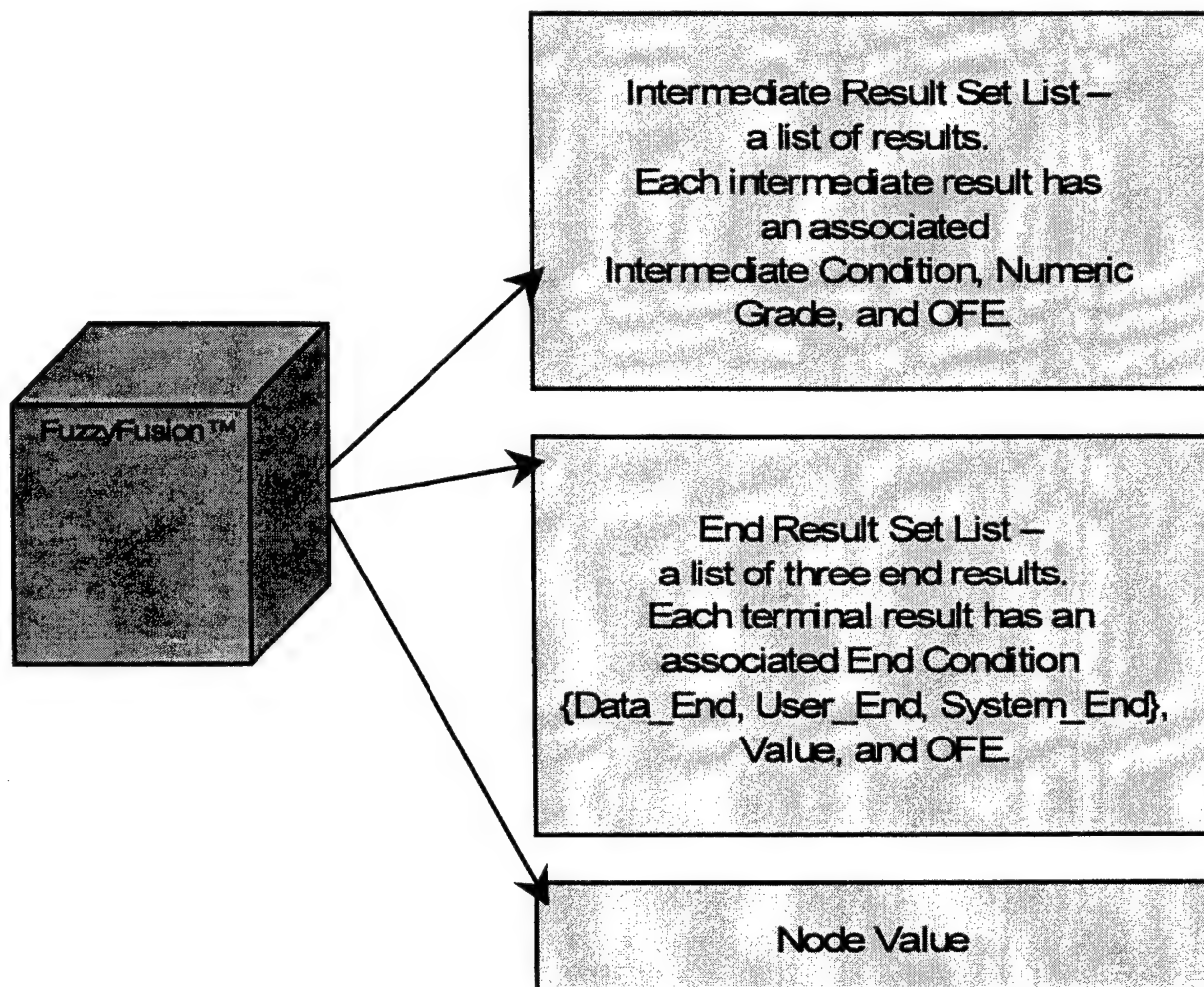


Figure 4.3-6 - Diagram of FuzzyFusion™ outputs

4.3.5.4 IDEA FuzzyFusion™ processing overview

The IDEA FuzzyFusion™ algorithm can be decomposed into nine steps.

1. List reachable Intermediate Conditions from Relevant Rules.
2. Calculate Numeric Grade for each reachable Intermediate Condition.
3. Calculate Belief for each reachable Intermediate Condition.
4. Calculate maximum OFE for each reachable Intermediate Condition.
5. Apply fuzzy set membership functions of Intermediate Conditions to generate fuzzy End Conditions.
6. Apply to Belief of the Intermediate Conditions.
7. De-fuzzify the fuzzy sets to obtain a value for each End Condition.
8. Apply fuzzy set characteristic functions to Intermediate Conditions' OFEs to generate maximum OFEs.
9. Calculate a Node Value using linearly weighted End Conditions' values and OFEs.

4.3.5.5 Algorithm details

4.3.5.5.1 STEP 1

The first step is to apply the Relevant Rules to the Pruned Hits List so as to obtain the reachable Intermediate Conditions. Conceptually, this step applies the knowledge base created by security engineers to the tool results. Because the tool results have been previously pruned by application of policy, this step is efficient.

4.3.5.5.2 STEP 2

The next step is to calculate the Numeric Grade for each reachable Intermediate Condition. This involves two sub-steps. Sub-step A requires calculation of the Trust T_{Ri} associated with Rule_i. Because there may be Hits produced from different tools, there may be various Trusts of those Hits. Sub-step A requires obtaining the mean of those trusts.

$$T_{Ri} = \text{arithmetic mean (Trusts of Hits that made the Rule a Relevant Rule).}$$

Figure 4.3-7 –Calculation of Numeric Grade, sub-step A

Sub-step B requires calculation of the Numeric Grade associated with an Intermediate Condition. Numeric Grade is the sum of the trusts of the Rules which resulted in the Intermediate Condition.

$$\text{Numeric Grade}_{\text{Intermediate Condition}} = \sum T_{Ri}$$

Figure 4.3-8 -Calculation of Numeric Grade, sub-step B

Numeric Grade characterizes how many Hits lead to the Intermediate Condition and how much they are trusted. It reflects the notion of “how big is the hole” in security.

4.3.5.5.3 STEP 3

For each reachable Intermediate Condition, define a fuzzy measure of the certainty (belief) for each Reachable Intermediate Condition, $\text{Belief}_{\text{Intermediate Condition}}$. In calculation of Belief, we use the T_{Ri} defined in Step 2, sub-step A.

$$\text{Belief}_{\text{Intermediate Condition}} = \max \{ T_{Ri} \}$$

Figure 4.3-9 – Calculation of Belief

This step uses fuzzy measurement theory, and standard fuzzy union of underlying beliefs. Calculation of Belief by using maximum can be interpreted as a pessimistic acceptance of tools: If a most trusted tool finds a vulnerability, we believe the Intermediate Condition to the extent of our trust in that tool. Often, but not always, Belief will be 1.

4.3.5.5.4 STEP 4

For each reachable Intermediate Condition, calculate the Opportunity for Exploitation, OFE, as the maximum of the Possibilities of the Hits which triggered rules leading to the Intermediate Condition.

$$\text{OFE}_i = \max\{\text{Possibility of each Hit mapped to that Intermediate Condition}\}$$

Figure 4.3-10 – Calculation of Opportunity for Exploitation.

OFE (opportunity for exploitation) characterizes the maximum opportunity that a vulnerability could be exploited. OFE ranges between [0, 1].

4.3.5.5.5 STEP 5

We selected three End Conditions, which are fuzzy sets named Data_End, User_End, and System_End. Each set is defined by fuzzy set membership functions which have been pre-defined by knowledge (security) engineers. Each fuzzy set is composed of Intermediate Conditions and their respective membership values. For example,

End Condition_i = {(Intermediate Condition₁, membership value₁) ...
(Intermediate Condition_n, membership value_n)}

Figure 4.3-11- Fuzzy membership function

Table 7.5-1 contains the fuzzy membership values use by the FuzzyFusion™ component.

- Account access
- Administrator access
- Backdoor access
- Data access
- File access
- Hardware access
- Hijack access
- Information access
- Network access
- Password access
- Privilege access
- Process access
- User access
- Compromised
- Encryption
- TBI

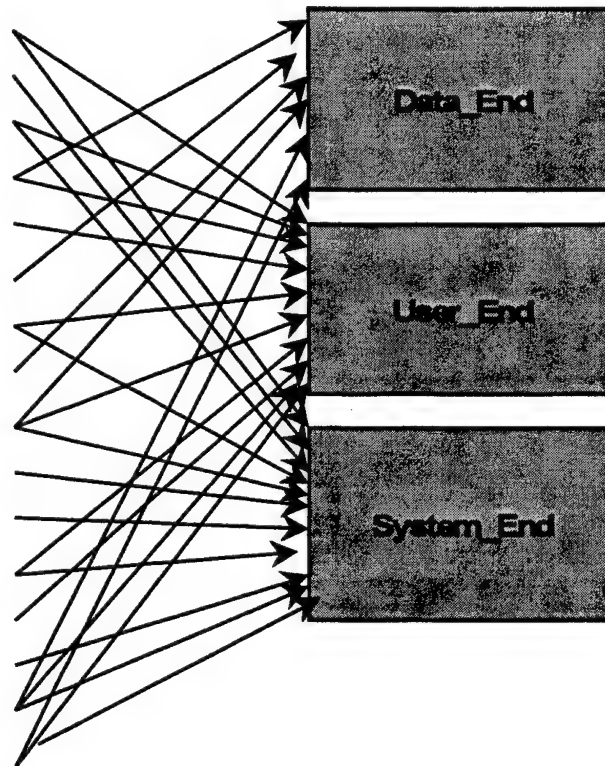


Figure 4.3-12 – Characteristic mappings of 16 Intermediate Conditions to End Condition fuzzy sets.

4.3.5.5.6 STEP 6

Because Intermediate Conditions are not crisp sets, but rather have fuzzy belief measures, we replace the definition in Step 5 with this more flexible definition of the fuzzy membership function.

$$\text{End Condition}_i = \{(\text{Intermediate Condition}_1, \text{membership value}_1 \cdot \text{belief}_n) \dots$$

$$(\text{Intermediate Condition}_n, \text{membership value}_n \cdot \text{belief}_n)\}$$

Figure 4.3-13 – Fuzzy membership function incorporating fuzzy belief measures

This definition lets us incorporate how much we believe in the tools into our End Condition.

4.3.5.5.7 STEP 7

The fuzzy membership function for each End Condition must be evaluated to be useful. We used standard fuzzy center of gravity ("COG") de-fuzzification to get values for the three fuzzy sets. E.g.

$$\text{Value}_{\text{Data_End}} = \text{COG}(\text{Data_End})$$

Figure 4.3-14 – Defuzzification method

Since the 16 Intermediate Conditions are not ordered, or even partially ordered, the COG calculation specializes to be the mean in current implementation. The Value is always between 0 and 1.

4.3.5.5.8 STEP 8

Next, we use standard fuzzy union of the OFE for the Intermediate Conditions participating in the membership function to get a OFE value for each of the three fuzzy sets. This is a standard application of the crisp characteristic function to an attribute of the fuzzy set. For example, for Data_End where the characteristic function is g

$$\text{OFE}_{\text{Data_End}} = \max(\text{PFE}_{\text{Intermediate Condition}}, \text{where } g(\text{Intermediate Condition}) = 1)$$

Figure 4.3-15 – Calculation of OFE for End Condition

OFE of the End Condition is a pessimistic view of the opportunity for exploitation found in the Intermediate Conditions composing that End Condition. In other words, if there is a high OFE anywhere, it is presumed that an attacker would choose that vulnerability to exploit. The OFE is always between 0 and 1.

4.3.5.5.9 STEP 9

Some users may desire a way to compare the overall security vulnerability of one node in a network against another node. In this final step, we calculate a numeric value for the node using linear weighting:

$$\begin{aligned} \text{Node Value} = & 0.167(\text{Value}_{\text{Data_End}} \cdot \text{POE}_{\text{Data_End}}) \\ & + 0.333 \cdot (\text{Value}_{\text{User_End}} \cdot \text{POE}_{\text{User_End}}) \\ & + 0.500 \cdot (\text{Value}_{\text{System_End}} \cdot \text{POE}_{\text{System_End}}) \end{aligned}$$

Figure 4.3-16 – Calculation of final node value

The Node Value is always between 0 and 1.

4.3.5.5.9.1 The divergence of IDEA FuzzyFusion™ from Stat Analyzer FuzzyFusion™

Stat Analyzer FuzzyFusion™ contained an inferencing process. The algorithm is fundamentally as shown in Figure 4.3-17.

In our review of that inferencing process, we made three observations. First, little inferencing actually occurred. Second, only two advanced states were reached by inferencing. Third, the advanced states were usually also reached directly by application of rules without inferencing. On reviewing the content of the knowledge engineers' rules, it appeared that almost all rules were stated giving a conclusion of the final state reachable, rather than giving atomic, small-step conclusions which could be chained together to reach conclusions about final states.

We concluded that we could omit the inferencing algorithm, and apply the Relevant Rules to the Pruned Hits without needing a Start_Condition. This required minor restatement of the rules.

Apply Relevant Rules to Pruned Hits and a given Start_Condition to find elevated Conditions.

do Apply Relevant Rules to Pruned Hits and elevated Conditions

while (new elevated Conditions were produced)

Output list of elevated Conditions as the Intermediate Conditions

Figure 4.3-17 – STAT® Analyzer FuzzyFusion™ Inferencing algorithm

4.3.5.6 Conclusions

The IDEA FuzzyFusion™ algorithm provides three levels of granularity for viewing the security condition of the node to which it is applied. Each granularity may be of use to system users who have different needs. The algorithm provides detailed information over 16 conditions, which is useful to users desiring to prioritize repair tasks. It provides summary information over three end conditions, which is useful to users needing to evaluate the specific vulnerability with respect to data, users, and the system. Finally, FuzzyFusion™ provides single, summarizing value for a node. The single value facilitates comparisons across nodes in a network.

4.4 The Proof-Of-Concept Prototype

This section describes the implementation and use of the FuzzyFusion™ component prototype.

Appendix D contains the instructions for the prototype FuzzyFusion™ component delivered with IDEA.

Section 4.4.1 is generated by Rational SoDA, a documentation tool that generates a Word document from Rational Rose.

4.4.1 LOGICAL ARCHITECTURE

4.4.1.1 Overview

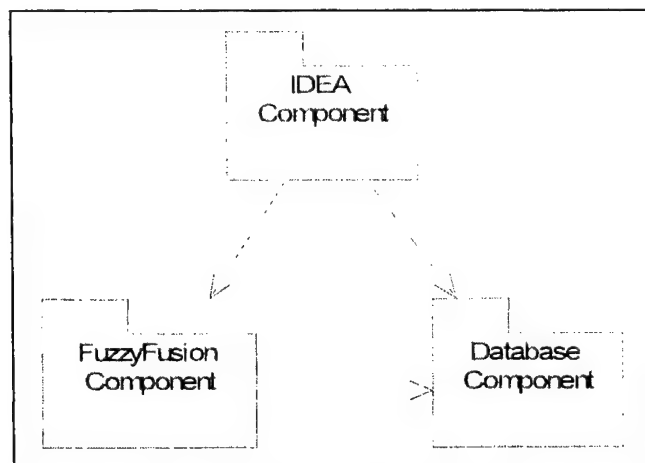


Figure 4.4-1: Component Layout

4.4.1.2 Class Structure

4.4.1.2.1 Overview

Name	Type	Description
m hIcon	HICON	

4.4.1.2.2.2 Operations

Name	Description
OnImportbtn	
OnStartbtn	
OnResetDBbtn	
OnBrowsebtn	
OnQueryDragIcon	
OnPaint	
OnInitDialog	
DoDataExchange	
CIdeaGUIDlg	

4.4.1.2.2.3IdeaAdoInterface

4.4.1.2.2.3.1 Attributes

4.4.1.2.2.3.2 Operations

Name	Description
disconnectFromDatabase	<pre> ***** NAME: disconnectFromDatabase /* /* TYPE: Function /* /* PURPOSE: Uses special pointer method "Release" to free memory used by the /* ADO Connection Object. /* ***** </pre>
connectToDatabase	<pre> ***** NAME: connectToDatabase /* /* TYPE: Function /* /* PURPOSE: Creates an instance of the ADO Connection Object. And opens the /* connection to the database. The reference it uses is a member /* class variable. /* ***** </pre>
openRecordset	<pre> ***** NAME: openRecordset /* /* TYPE: Function - Private /* /* PURPOSE: Opens an ADO Recordset using a passed in reference to a Recordset /* and a passed in query string. /* ***** </pre>
~IdeaAdoInterface	
IdeaAdoInterface	Constructor and Destructor
executeSelect	<pre> ***** NAME: executeSelect </pre>

	<pre> /* ***** clearStartCondition NAME: clearStartCondition /* /* TYPE: Function /* /* PURPOSE: Reset the start condition stored in the database. /* ***** </pre>
clearAll	<pre> ***** NAME: clearAll /* /* TYPE: Function /* /* PURPOSE: Reset the entire database. /* ***** </pre>
fuse	<pre> ***** NAME: fuse /* /* TYPE: Function /* /* PURPOSE: Calls the fuse method of the Fuzzy Fusion API. /* ***** </pre>
importData	<pre> ***** NAME: importData /* /* TYPE: Function /* /* PURPOSE: Extract data from an input file in XML format using an MS XML /* parser. /* ***** </pre>
~IdeaApi	
IdeaApi	

4.4.1.2.2.5IdeaDatabaseApi

4.4.1.2.2.5.1 Attributes

4.4.1.2.2.5.2 Operations

Name	Description
~IdeaDatabaseApi	
IdeaDatabaseApi	Constructor and Destructor
executeSelect	<pre> ***** NAME: executeSelect /* /* TYPE: Pure Virtual Function /* /* PURPOSE: Provides an abstract interface. This function shall be defined /* to execute a SELECT query. A boolean value shall be returned to /* signify the success or failure of the query execution. The value /* parameters returnRS and numRows shall be populated with data /* returned from the query execution. /* ***** </pre>
executeModify	<pre> ***** NAME: executeModify /* </pre>

	<pre> /* TYPE: Pure Virtual Function /* /* PURPOSE: Provides an abstract interface. This function shall be defined /* to execute an UPDATE, INSERT, or DELETE query. A boolean value /* shall be returned to signify the success or failure of the /* query execution. /* /* /***** </pre>
--	--

4.4.1.2.2.6IdeaDataManager

4.4.1.2.2.6.1 Attributes

4.4.1.2.2.6.2 Operations

Name	Description
insertGrade	<pre> ***** NAME: insertGrade /* /* TYPE: Function /* /* PURPOSE: Executes a SQL statement to insert a grade into the database. /* /***** </pre>
resetVulnerabilities	<pre> ***** NAME: resetVulnerabilities /* /* TYPE: Function /* /* PURPOSE: Executes a SQL statement to delete all vulnerabilities from the /* database. /* /***** </pre>
resetPolicies	<pre> ***** NAME: resetPolicies /* /* TYPE: Function /* /* PURPOSE: Executes a SQL statement to delete all policies from the /* database. /* /***** </pre>
resetStartCondition	<pre> ***** NAME: resetStartCondition /* /* TYPE: Function /* /* PURPOSE: Call a stroed procedure to reset the start condition. /* /***** </pre>
initializedDB	<pre> ***** NAME: intializedDB /* /* TYPE: Function /* /* PURPOSE: Calls other member functions to in order to completely reset the /* database. /* /***** </pre>
insertStartCondition	<pre> ***** NAME: insertStartCondition /* </pre>

	<pre> /* TYPE: Function /* /* PURPOSE: Creates a SQL statment to insert a start condition into the DB. /* /***** </pre>
insertVulnerability	<pre> ***** NAME: insertVulnerability /* /* TYPE: Function /* /* PURPOSE: Creates a SQL statment to insert a vulnerability into the database. /* /***** </pre>
insertVulnerability	
insertPolicy	<pre> ***** NAME: insertPolicy /* /* TYPE: Function /* /* PURPOSE: Creates a SQL statment to insert a policy into the database. /* /***** </pre>
~IdeaDataManager	
IdeaDataManager	

4.4.1.2.2.7IdeaFloatType

4.4.1.2.2.7.1 Attributes

Name	Type	Description
fltValue	float	

4.4.1.2.2.7.2 Operations

Name	Description
equals	<pre> ***** NAME: equals /* /* TYPE: Function /* /* PURPOSE: Compares another IdeaGenericDataType object with itself and returns /* true if they match, false if they don't. /* /***** </pre>
toString	<pre> ***** NAME: toString /* /* TYPE: Function /* /* PURPOSE: Definition of the pure virtual function inherited from GenericDatatype. This function is defined to cast the value stored in the member variable fltValue into a string. The string value is then returned . /* /***** </pre>

floatVal	<pre> ***** NAME: floatVal /* /* TYPE: Function /* /* PURPOSE: Returns the value of the member variable fltValue. /* /* ***** </pre>
~IdeaFloatType	
IdeaFloatType	Constructor and Destructor

4.4.1.2.2.8IdeaFuzzyDataManager

4.4.1.2.2.8.1 Attributes

4.4.1.2.2.8.2 Operations

Name	Description
setTools	
getPrunedHitsFromDB	
getPrunedRulesFromDB	Gets start condition for the node from the database. Returns 0 for success; ____ for other errors
getStartConditionFromDB	
~IdeaFuzzyDataManager	
IdeaFuzzyDataManager	

4.4.1.2.2.9IdeaFuzzyFusion

4.4.1.2.2.9.1 Attributes

4.4.1.2.2.9.2 Operations

Name	Description
findMaximalEndCondition	addToResults adds row to result set for the specified post condition. Returns 0 if successful
addToResults	getPOEOfRule returns the probability of exploitation of a rule based on the probability of the hits relied on by that rule. Returns 0 if successful
getPOEOfRule	getTrustOfRule returns the trust for a rule based on the trust of the hits relied on by that rule. Returns 0 if successful
getTrustOfRule	fillPostVector generates vector of strings containing distinct post conditions. Returns 0 if successful
fillPostVector	
fuse	Methods
fuse	
getEndCondition	Gets results by reference. Returns 0 if successful, 1 if failure.
getResults	Accessors
~IdeaFuzzyFusion	
IdeaFuzzyFusion	Constructors/destructor
IdeaFuzzyFusion	

4.4.1.2.2.10 IdeaFuzzyFusionApi

```
typedef vector<IdeaHit> HitVector;
typedef vector<IdeaRule> RuleVector;
typedef vector<IdeaResult> ResultVector;
typedef vector<IdeaStringType> ToolVector;
typedef vector<GradeConversionTable> ConversionTableVector;
```

4.4.1.2.2.10.1 Attributes

4.4.1.2.2.10.2 Operations

Name	Description
setTools	
getPrunedHitsFromDB	Gets the pruned rules from the database. Invokes IdeaFuzzyDataManager::getPrunedRulesFromDB Returns 0 on success.
getPrunedRulesFromDB	Gets start condition for the node from the database Invokes IdeaFuzzyDataManager::getStartConditionFromDB Returns 0 on success.
getStartConditionFromDB	Mutators int setStartCondition(const string aStartCondition); // Sets the start condition to e.g. data, user, system. // Returns 0 on success.
fuse	Methods
getEndCondition	Accessors. All return 0 on success.
~IdeaFuzzyFusionApi	
IdeaFuzzyFusionApi	Constructors/destructor

4.4.1.2.2.11 IdeaGenericDataType

4.4.1.2.2.11.1 Attributes

4.4.1.2.2.11.2 Operations

Name	Description
equals	***** NAME: equals /* /* TYPE: Virtual Function /* /* PURPOSE: Compares another IdeaGenericDataType object with itself and returns /* true if they match, false if they don't. /* /* *****
setType	***** NAME: setType /* /* TYPE: Function /* /* PURPOSE: Set the value of the member variable dataTypeEnum. This function /* has a protected scope so that only a derived class can call it. /* *****

toString	<pre> /***** ***** NAME: toString /* /* TYPE: Pure Virtual Function /* /* PURPOSE: Provides an abstract interface. This function shall be defined /* to cast the stored value into a string. The string value shall /* then be returned . /* /* *****/ </pre>
getType	<pre> /***** ***** NAME: getType /* /* TYPE: Function /* /* PURPOSE: Returns the value of the member variable dataTypeEnum. /* /* *****/ </pre>
~IdeaGenericDataType	
IdeaGenericDataType	Constructor and Destructor

4.4.1.2.2.12 IdeaHit

4.4.1.2.2.12.1 Attributes

Name	Type	Description
theTrust	float	
theProbability	float	

4.4.1.2.2.12.2 Operations

Name	Description
setHit	Mutators
getToolName	
getNum	
getProbability	
getTrust	Accessors - all return 0 on success
~IdeaHit	
IdeaHit	Constructors/Destructor
IdeaHit	

4.4.1.2.2.13 IdealIntegerType

4.4.1.2.2.13.1 Attributes

Name	Type	Description
intValue	int	

4.4.1.2.2.13.2 Operations

Name	Description
equals	<pre> NAME: equals /* /* TYPE: Function /* /* PURPOSE: Compares another IdeaGenericDataType object with itself and returns /* true if they match, false if they don't. /* /* </pre>
intVal	<pre> NAME: intVal /* /* TYPE: Function /* /* PURPOSE: Returns the value of the member variable intValue. /* /* </pre>
toString	<pre> NAME: toString /* /* TYPE: Function /* /* PURPOSE: Definition of the pure virtual function inherited from /* GenericDatatype. This function is defined to cast the value /* stored in the member variable intValue into a string. The string /* value is then returned . /* /* </pre>
~IdeaIntegerType	
IdeaIntegerType	Constructor and Destructor

4.4.1.2.2.14 IdeaResult

4.4.1.2.2.14.1 Attributes

Name	Type	Description
theNumericGrade	float	
thePOE	float	

4.4.1.2.2.14.2 Operations

Name	Description
setResult	Mutator
getCondition	
getNumericGrade	
getPOE	Accessors. All return 0 on success.
~IdeaResult	
IdeaResult	Constructors/destructor
IdeaResult	

4.4.1.2.2.15 IdeaRule

4.4.1.2.2.15.1 Attributes

4.4.1.2.2.15.2 Operations

Name	Description
setRule	Mutator
getRuleID	
getPre	
getPost	
getNum	accessors Return 0 on success
~IdeaRule	
IdeaRule	constructors/destructor
IdeaRule	

4.4.1.2.2.16 IdeaStringType

4.4.1.2.2.16.1 Attributes

Name	Type	Description
theStringLength	int	
strValue	char*	

4.4.1.2.2.16.2 Operations

Name	Description
IdeaStringType::equals	***** ***** NAME: operator == /* /* TYPE: Equality test Operator /* /* PURPOSE: Returns true if values are equal, false if not. /* /***** *****
IdeaStringType::operator= or=	***** ***** NAME: operator = /* /* TYPE: Assignment Operator /* /* PURPOSE: // Note: C++ requires that = can be overloaded only as nonstatic member function. /* /***** *****

IdeaStringType::getStringLength	<pre> ***** NAME: getStringLength /* /* TYPE: Function /* /* PURPOSE: Returns length of the string. /* /***** </pre>
IdeaStringType::toString	<pre> ***** NAME: toString /* /* TYPE: Function /* /* PURPOSE: Definition of the pure virtual function inherited from /* GenericDatatype. This function is defined to return the value /* stored in the member variable strValue. /* /***** </pre>
IdeaStringType::stringValue	<pre> ***** NAME: stringValue /* /* TYPE: Function /* /* PURPOSE: Returns the value of the member variable strValue as a char*. /* /***** </pre>
IdeaStringType::~~IdeaStringType	
IdeaStringType::IdeaStringType	Constructor and Destructor
IdeaStringType::IdeaStringType	
IdeaStringType::IdeaStringType	Copy constructor to properly init new instances of the class IdeaStringType

4.4.1.3 IDEA Component

4.4.1.3.1 Overview

The IDEA Component is responsible for providing a complete interface to client applications. The IdeaApi class contains the component's interface operations.

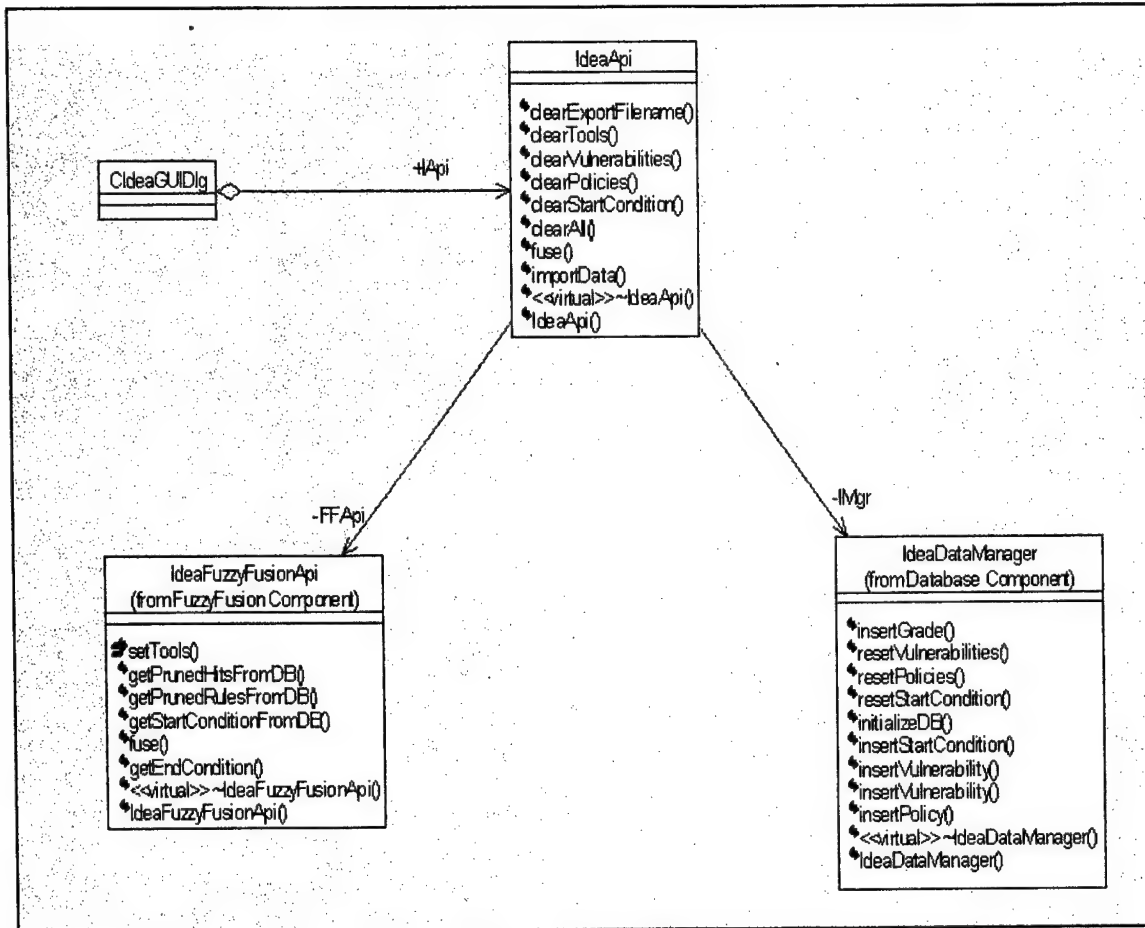


Figure 4.4-3: Main

4.4.1.3.2 Classes

4.4.1.3.2.1 IdeaApi (NormalClass)

4.4.1.3.2.2 CideaGUIDlg (NormalClass)

////////////////////////////////////

CideaGUIDlg dialog

4.4.1.4 FuzzyFusion Component

4.4.1.4.1 Overview

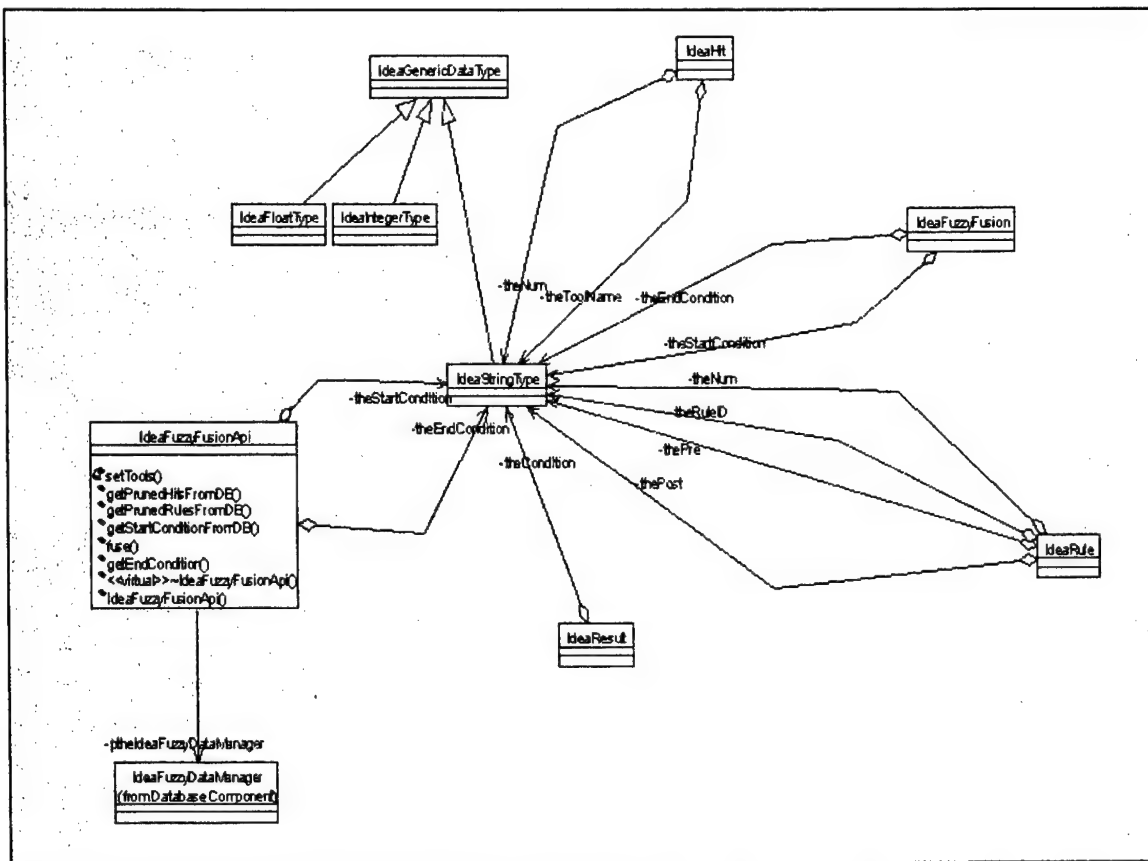


Figure 4.4-4: Main

4.4.1.4.2 Classes

4.4.1.4.2.1 IdeaFuzzyFusionApi (NormalClass)

```
typedef vector<IdeaHit> HitVector;
typedef vector<IdeaRule> RuleVector;
typedef vector<IdeaResult> ResultVector;
typedef vector<IdeaStringType> ToolVector;
typedef vector<GradeConversionTable> ConversionTableVector;
```

4.4.1.4.2.2 *IdeaFloatType* (NormalClass)

4.4.1.4.2.3 *IdealIntegerType* (NormalClass)

4.4.1.4.2.4 *IdeaGenericDataType* (NormalClass)

4.4.1.4.2.5 IdeaHit (NormalClass)

4.4.1.4.2.6 IdeaFuzzyFusion (NormalClass)

4.4.1.4.2.7 IdeaRule (NormalClass)

4.4.1.4.2.8 IdeaResult (NormalClass)

4.4.1.4.2.9 IdeaStringType (NormalClass)

4.4.1.5 Database Component

4.4.1.5.1 Overview

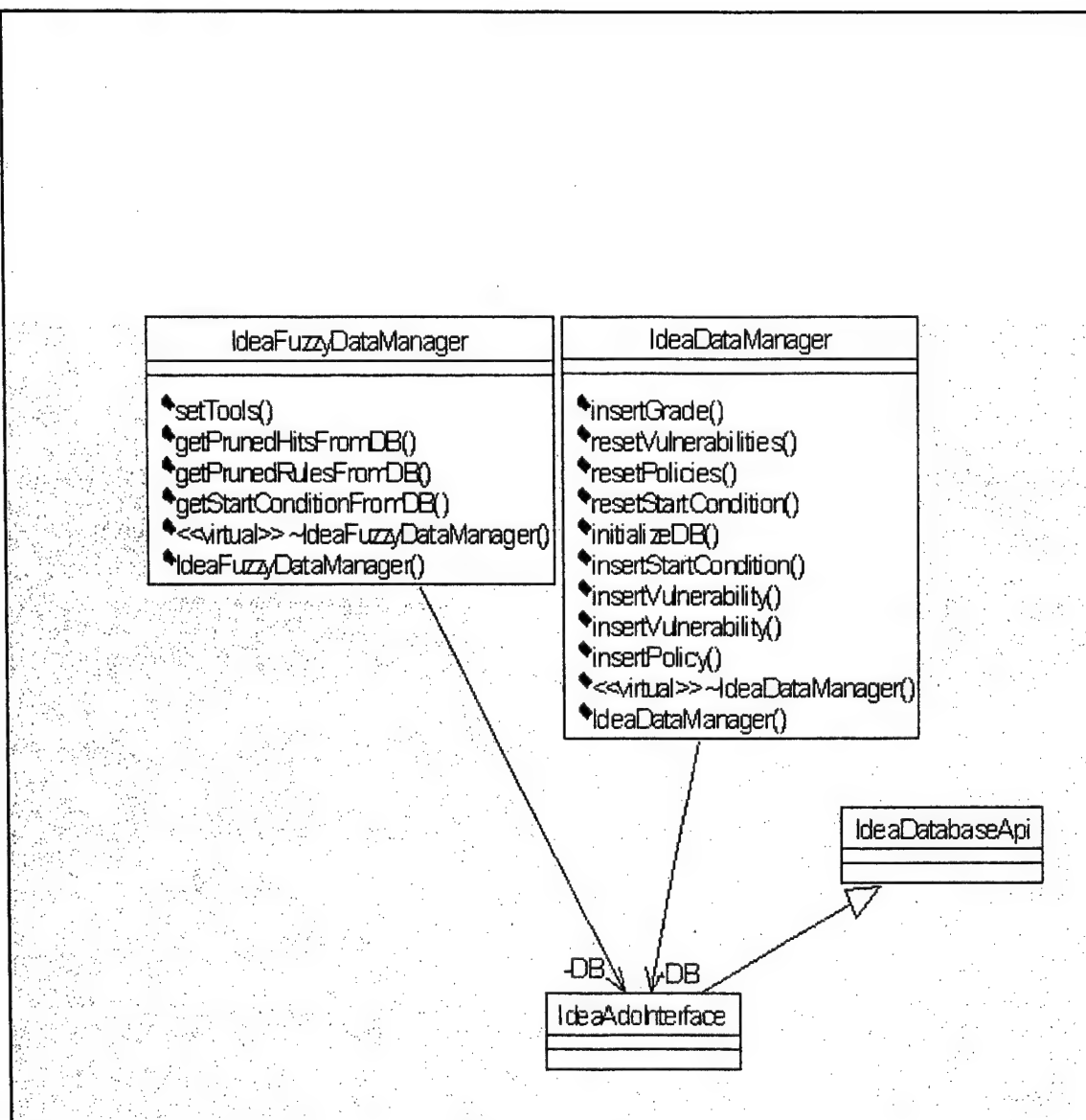


Figure 4.4-5: Main

4.4.1.5.2 Classes

4.4.1.5.2.1 *IdeaDatabaseApi (NormalClass)*

4.4.1.5.2.2 *IdeaAdolInterface (NormalClass)*

4.4.1.5.2.3 *IdeaDataManager (NormalClass)*

4.4.1.5.2.4 *IdeaFuzzyDataManager (NormalClass)*

4.5 Testing and Evaluation

4.5.1 Analysis of results

Along with generating the intermediate conditions (Vulnerability Category, Grade, and OFE), the FuzzyFusion™ component generates three additional measures:

- 1) The values for three End Conditions (Data_End value, User_End value, and System_End value),
- 2) End value,
- 3) Terminal Condition (one of Data, User, System). This value is current unimplemented, but would be the greater of the three – Data, User, System).

The seventeen intermediate conditions lead to the three fuzzy sets: Data_End, User_End, and System_End. The values of the three fuzzy sets are calculated using a Center Of Gravity (COG) function. Using COG, IDEA calculates the defuzzified values for the End Conditions using the Trust and the vulnerability category contributions as defined by the fuzzy membership functions.

End value is calculated directly from Data_End, User_End, and System_End. The Terminal Condition, although not physically implemented, would be the larger of the three fuzzy sets: Data_End, User_End, or System_End.

IDEA used test data generated at the STAT® Analyzer test facility. This data is STAT® Scanner vulnerability results from STAT® Analyzer's test machines.

The following sections discuss the results of applying FuzzyFusion™ technology to the test data.

4.5.2 Quality of Results

The measure of quality of data fusion techniques can be problematic. It is often difficult to identify what constitutes ground truth, and even more difficult to obtain that ground truth for comparison to algorithm results. If ground truth is not obtainable, it is useful to select a "strawman" for testing. The results of IDEA FuzzyFusion™ are tested for validity against the strawman, because a ground truth was not obtainable.

4.5.2.1 Ground truth

We attempted to obtain ground truth by consulting four security engineers. We asked each individually to review scan information for a small network and to report on "the overall security vulnerability" of each node at the time of each scan. We provided STAT® Scanner results of 44 scans on twelve individual machines. Each machine had been scanned two to five times. The results from STAT® Scanner include a detailed description of the vulnerability, as well as a categorization of each vulnerability as "high severity," "medium severity," or "low severity."

After beginning the review, the security engineers reported that because STAT® Scanner located between 60 and 190 individual vulnerabilities with each scan of a node, it was not possible to synthesize those vulnerabilities and reach a conclusion on "the overall security vulnerability" of each node at the time of each scan. The security engineers then met and suggested a heuristic that if any "high" severity vulnerability was found on a node, it should

be classified as highly vulnerable. We deemed this heuristic not useful. Each node in the network had at least one "high" severity vulnerability. The heuristic would not allow useful discrimination among the vulnerability level of the machines.

4.5.2.2 Strawman

We chose to use the distribution of the vulnerabilities as the basis for a strawman. If the number of "low severity" vulnerabilities reported by STAT® Scanner was within one standard deviation of the mean, the node would be labeled "medium vulnerable." If the number of vulnerabilities was below minus one standard deviation, the node would be labeled "low vulnerable;" if above plus one standard deviation, it would be labeled "high vulnerable."

Figure 4.5-1 shows the distribution of vulnerabilities for 44 scans. The correlation of the counts of low, medium, and high vulnerabilities is visually apparent. Scatter graphs confirm the correlation.

Using the strawman having a mean of 106, and a standard deviation of 17, we identified four scans where the count of low vulnerabilities was below 89, and one scan where the count of low vulnerabilities was above 123. The other 39 scans were labeled medium vulnerable.

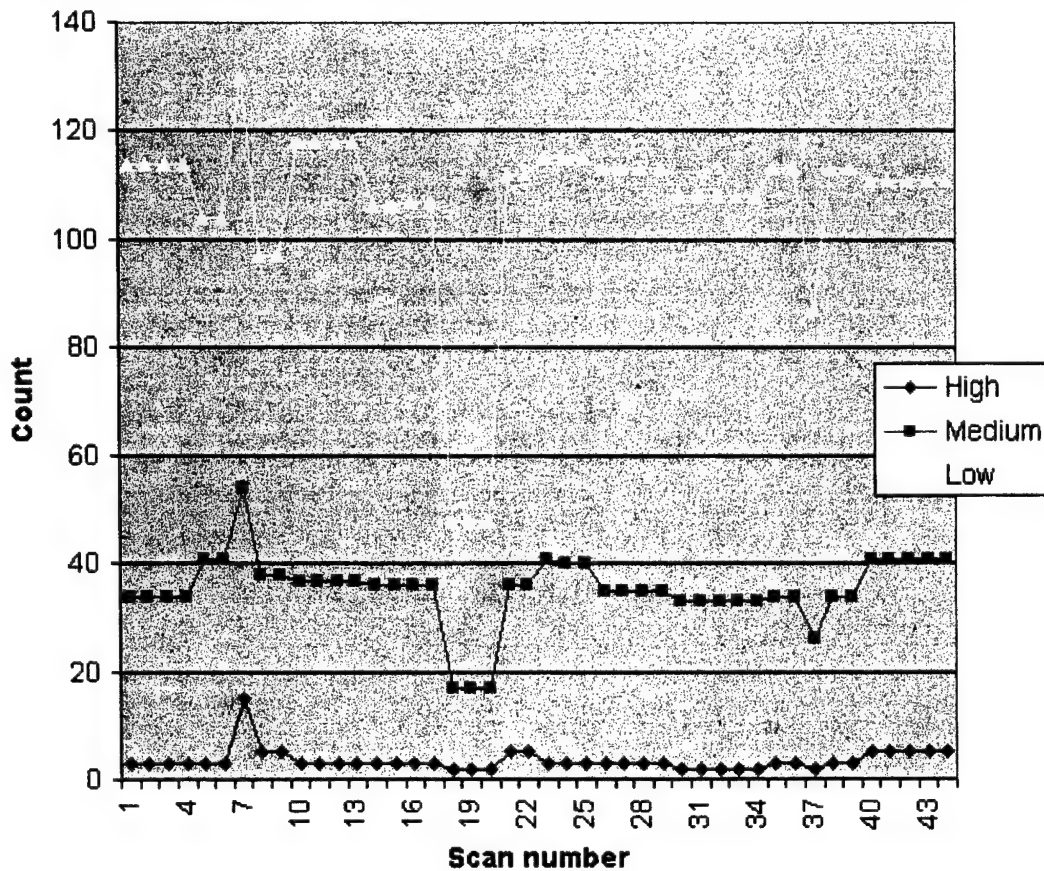


Figure 4.5-1 - Distribution of severity levels reported by STAT® Scanner during forty-four scans of twelve nodes.

The graph shows the vulnerability count found by STAT® Scanner across 44 scans. Note that the bottom value represents the vulnerabilities with the greatest importance.

4.5.3 Comparison of results

4.5.3.1 Defuzzification of End Conditions Method 1

The values the FuzzyFusion™ component returns for the End Conditions are based upon the fuzzy membership functions and the user-provided value of vulnerability Trust. Figure 4.5-2 shows the results of the FuzzyFusion™ component as implemented on IDEA.

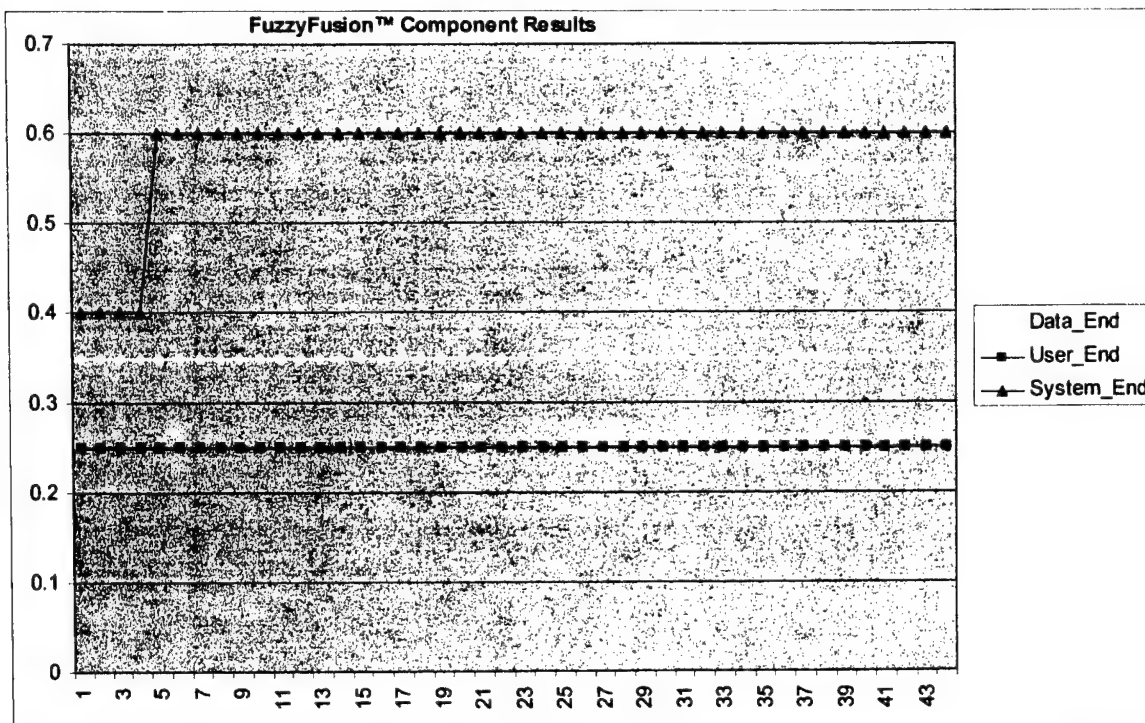


Figure 4.5-2 – FuzzyFusion™ Component Results

Analysis of the figure yields inconclusive information when the Trust provided for each vulnerability is 1 – meaning that we fully trust the vulnerability to be accurate. The assumption that Trust would normally be 1 is based on STAT® experience that only rarely is vulnerability information incorrect. With the Trust of the input data set to 1, the FuzzyFusion™ algorithm is simply adding the fuzzy membership contributions for the calculations of the End Conditions.

For our next step of data analysis we imported the FuzzyFusion™ component results into an Excel spreadsheet. Using Excel, we researched various defuzzification algorithms for generating values for the end conditions (Data_End, User_End, and System_End).

4.5.3.2 Defuzzification of End Conditions Method 2

Our first approach was to use the intermediate results for Grade and OFE along with the fuzzy membership functions. The values for the End Conditions are strikingly similar to that in STAT® Scanner's output (see Figure 4.5-1).

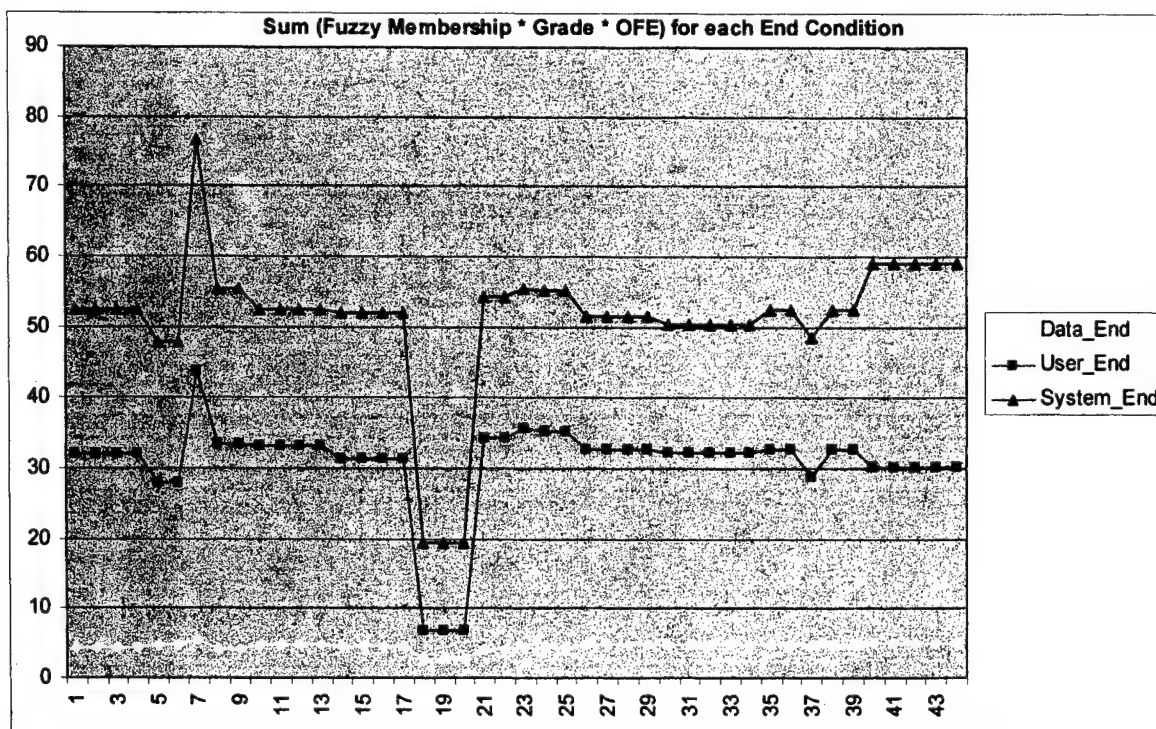


Figure 4.5-3 End Conditions using Fuzzy Membership Contributions and Grade and OFE

Unlike the STAT® Scanner output in Figure 4.5-1, Figure 4.5-3 ranks the most pressing vulnerabilities highest in the graph. As does STAT® Scanner, FuzzyFusion™ concludes that node 8 contains the greatest system-level vulnerabilities. However, in contrast to STAT® Scan's results, FuzzyFusion™'s results show one node (node 8) as having the highest exposure (approximate value of 76).

The distinguishing factor in using Opportunities for Exploitation in the FuzzyFusion™ calculations is that the outcome may be tailored to the nature of the network or node. If it is not possible to gain physical access to a machine, several exploitation opportunities would not be possible. For example, exploiting the fact that the clipboard's contents are available at the login screen is not possible without physical access to the machine. So, in addition to Security Policy – which FuzzyFusion™ uses only to tailor its vulnerability assessment, OFE is very useful in fully understanding the vulnerability for a node.

To show how OFE is useful in vulnerability assessment, Figure 4.5-4 graphs identical vulnerability information, Grade, Trust, but minimizes the system access OFEs. In the figure, Data_End ranks highest for the same machine.

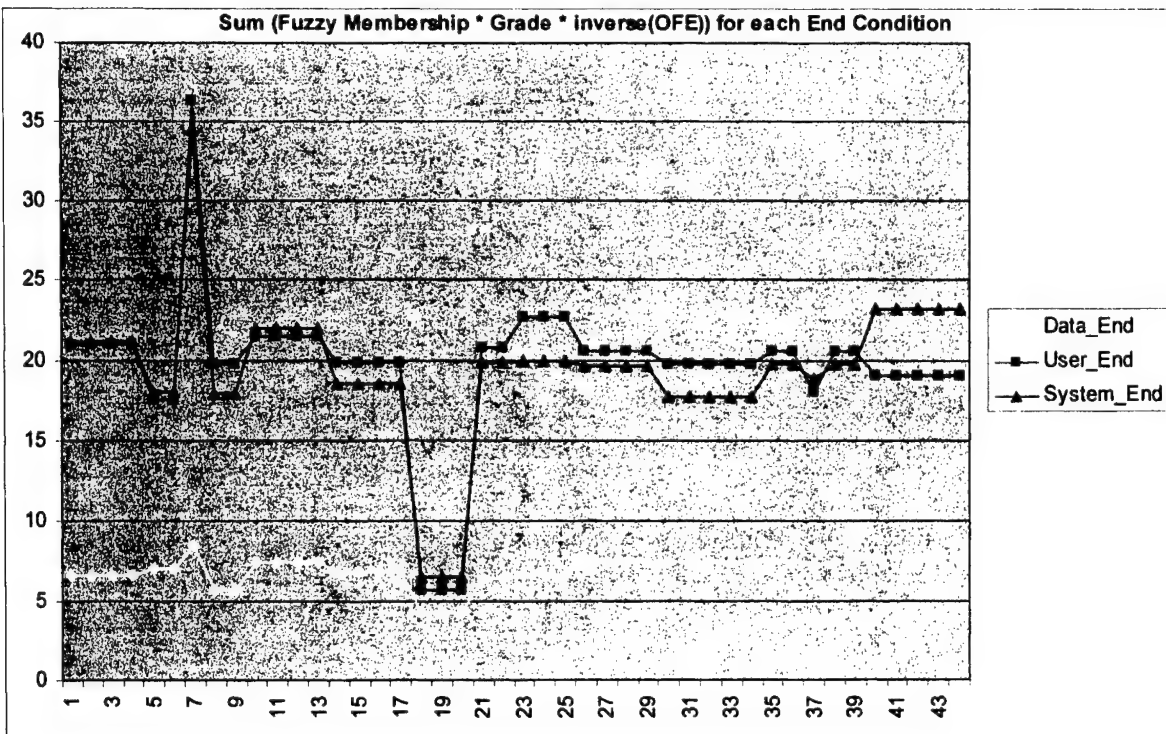


Figure 4.5-4 – Modifying OFE

4.5.3.3 Defuzzification of End Conditions Method 3

In the previous section Grade was used in the analysis. This is fine for a single tool, however vulnerability assessment across several tools (for a single node) would produce varied results. As tool's overlap in their ability to detect vulnerabilities (and as tool maturity differs), a vulnerability detected by one tool may not be detected by another. Grade, being a calculation dependent on the number of vulnerabilities, will skew results where tool-detected vulnerabilities overlap.

This analysis method removed Grade from the calculations of the values of the End Conditions. In Figure 4.5-5 we see that removing Grade slightly affected the trend of the result. In both figures (Figure 4.5-5 and Figure 4.5-1) the machine most significantly vulnerable is obvious.

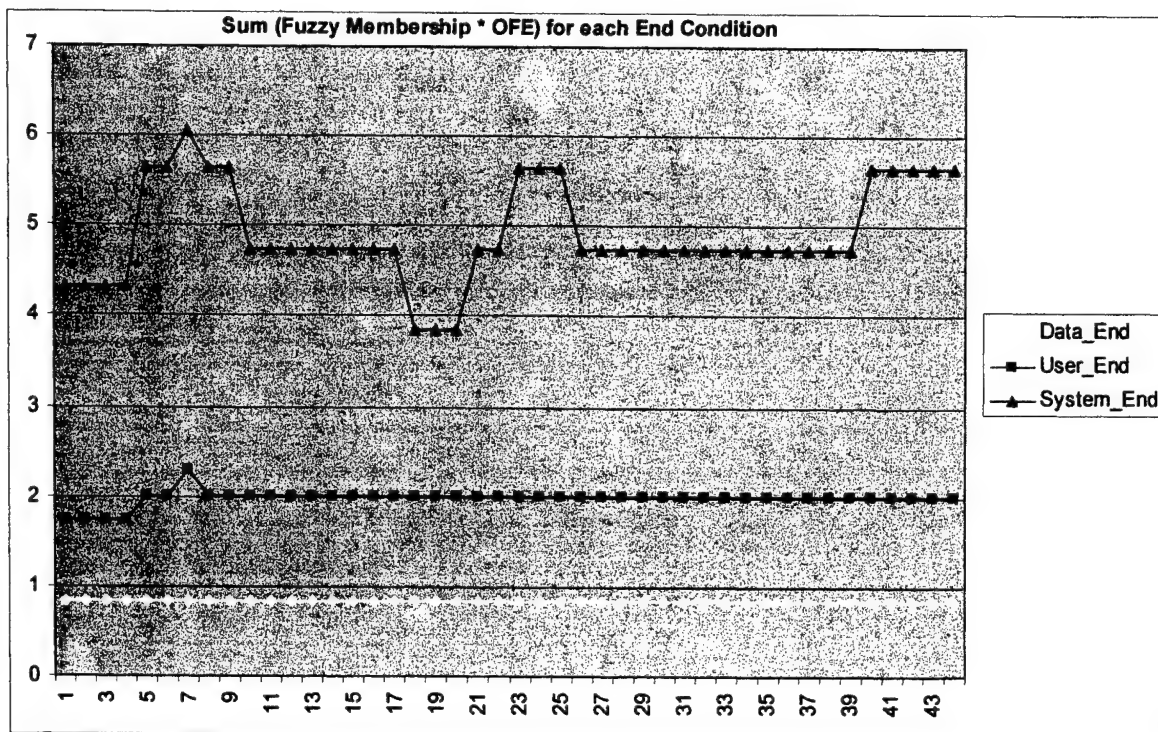


Figure 4.5-5 – End Conditions using Fuzzy Membership Contributions and OFE

Also, as in the previous analysis method, minimizing OFE for system access produces notable results. The same network node is again identified as being the most vulnerable node in the network.

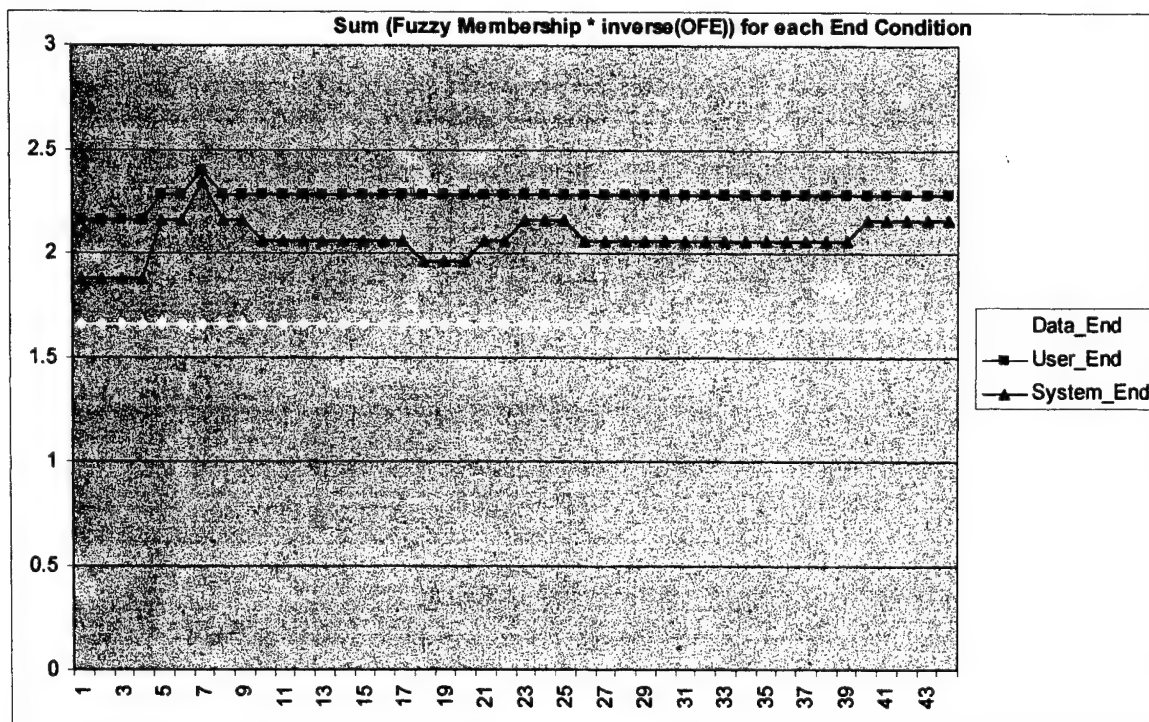


Figure 4.5-6 – Modifying OFE

4.5.3.4 Discussion of Results

The lack of discriminating results in Method 1 is a major drawback. Not being able to pinpoint or distinguish the more vulnerable nodes from the less vulnerable ones, the system administrator will be helpless.

Method 2 and 3 look very promising in that they are not only strikingly similar to the straight ranking of STAT® Scanner, but that they are able to distinguish individual nodes to fix. Since Method 3 supports multi-tool analysis it is the method of choice for FuzzyFusion™.

4.6 Research into Natural Language Processing

Towards the end of the IDEA effort, Kevin Pratt researched the feasibility of using Natural Language Processing (NLP) analysis on STAT® Scanner, ISS, and CyberCop vulnerability text. The results were very promising – NLP finds the proper category in more than 80% of the test examples.

4.6.1 Discussion of Research

Security engineers write descriptions of each of the vulnerabilities found by tools. They then spend substantial time assigning the vulnerabilities to various sets of categories for use in further analysis.

We investigated whether a program could be developed which would learn from the prior descriptions and their categories so that future descriptions could be automatically assigned to the appropriate categories, or in the alternative, so that a "short list" of likely category assignments could be offered to security engineers so as to speed their categorization task.

The descriptions are between 15 and 300 words in length, and are written in ordinary English by security engineers. The categories are phrases of one to four words.

We proceeded in two steps. The first step was to distill the descriptions. The global vocabulary was over 2500 words. We removed words deemed to have little content. Our list of 180 words to remove included common prepositions, adverbs, pronouns, and non-transitive verb forms, for example, "inside," "only," "they," and "being." We also removed words of one or two letters, for example "a," and "at." We next applied a heuristic to obtain roots from standard forms of plural nouns, past tenses and participles of verbs, and adverbs. We did not attempt to define roots for irregular verbs, nor did we attempt to define synonyms. We discarded all duplicate words. By distilling, we condensed the global vocabulary used in the descriptions from about 2500 words to about 1500 words. We did not parse the descriptions, nor did we attempt to establish the relations between words within each description or phrases in the descriptions.

We used the identical method to distill training example descriptions and testing example descriptions.

The second step was to apply learning algorithms. Learning involves training, followed by testing. We used various data sets, ranging from 450 training examples and three categories, to 2000 training examples and 150 categories. With each data set we tested on approximately 100 randomly selected new test examples and compared the categories selected by the learning algorithms with the actual categories specified by the security engineers in order to determine the effectiveness of each approach.

One standard representation of multi-featured data for use with learning algorithms is a feature vector. We constructed a data structure which held a schema describing each feature by name, and by type. Specifically, the schema used was an alphabetized list of the distilled vocabulary, and the type of each schema field was a C++ string. We stored each description as a binary valued feature vector of the vocabulary. Thus, with 450 examples and a vocabulary of 1500 words, we would have a table of 450 rows and 1500 columns containing 450 x 1500 elements. A 1 in the element meant that the word at that column of the schema was present in the description; a 0 meant it was absent. We also stored the category that had been assigned to the example. We constructed additional tables to hold information summarizing the category

The learning algorithms were these:

A For each category, make a list of all words that were found in the descriptions of the category. Count how many times the words in a test example were found in the list. Conclude that the list with the most matches indicates the best category for the test example.

For algorithms B and C, we calculated a past probability vector for each category. The vector has a value at every field which is the number of occurrences of a word divided by the total occurrences of the word in all categories.

B When a word in a test example was found in the past probability vector, add the value of the past probability vector at that feature to a total. Conclude that the category with the highest total indicates the best category for the test example.

C When a word in a test example was found in the past probability vector, add the value of the past probability vector at that feature to a total, and keep track of how many words had been found in that past probability vector. Conclude that the category with the highest average indicates the best category for the test example. We expected that this would work better than B because in B the total would be higher just because more words were found. Using the average would be less biased.

For the next algorithms, we always calculated a centroid vector for each category. The centroid has a value at every field which is the number of occurrences of a word divided by the maximum number of possible occurrences of the word in a category. Use of a centroid vector to characterize a category is a standard procedure used in clustering and learning.

D Measure the distance between the test example vector and the centroid of a category. The shortest distance indicates the best category for the test example. The distance is the sum of the distances at each feature.

E Measure the Euclidean distance between the test example vector and the centroid of a category. The shortest distance indicates the best category for the test example. Euclidean distance is the square root of the sum of the squares of the distances. It is a standard method in geometric space which has often been extended to feature vector analysis. Euclidean distance accentuates the distances where features are least similar.

F Treat the test example vector as a "mask" when applying algorithm D. The effect is to ignore the hundreds of words where there is no match. Instead, we look only at the distances for words existing in the example. We expected that this would work better than D because the "noise" from the features present in the centroid, but absent in the test example, would be ignored.

G Treat the test example vector as a "mask" when applying algorithm E. The effect is to ignore the hundreds of words where there is no match. Instead, we look only at the distances for words existing in the example. We expected that this also would work better than E.

H We began work on two additional algorithms. One would use a decision tree which selected features based on information theory, and pruned with chi-squared pruning, to be applied on a masked features. The other would apply Euclidean distance to normalized feature values. We did not complete these algorithms.

Results:

Random selection would result in a correct categorization in the inverse of the number of categories. For example, with 40 categories, random selection would find the correct category for 2.5% of the test examples.

We show results for the seven implemented algorithms in Table 4.6-1. We show the percentage of times the correct choice was the first choice, second choice, and third choice. We also show the percentage from random selection. In each instance, 80% of the examples were used for training, and 20% for testing.

Table 4.6-1 – Results from application of five learning algorithms.

Algorithm	550 Examples 38 Categories			2128 Examples 4 Categories			2141 Examples 9 Categories			2141 Examples 217 Categories		
	1st	2nd	3d	1st	2nd	3d	1st	2nd	3d	1st	2nd	3d
Random	2.6	2.6	2.6	25	25	25	11	11	11	0.5	0.5	0.5
A	66	13	3	52	29	18	60	14	11	43	12	6
B	61	17	5	59	28	12	63	11	6	41	11	9
C	60	17	5	62	25	13	64	13	6	45	12	6
D	41	12	1	22	15	16	17	9	8	21	7	7
E	59	18	5	62	25	12	65	13	6	45	12	8
F	41	12	1	22	15	16	65	13	6	21	7	7
G	38	19	14	42	31	24	65	13	6	32	8	7

We conclude that algorithms A, B, C and E perform well, far exceeding random. Tuning of the algorithms, and elaboration of the other algorithms mentioned above may be important future work. We expect that security engineers may also disagree about categorizations at times. We did not test how security engineers would have performed given the same test examples.

5 Conclusions

The IDEA research has resulted in a proof-of-concept prototype demonstrating a comprehensive vulnerability analysis based on vulnerability scanner data. Users of the component have a simple, open interface available for performing vulnerability assessment for a single node using multiple tool results.

The following sections summarize the conclusions that can be drawn from the IDEA effort. Next, a number of improvements to the IDEA prototype have been identified. These improvements would enhance the usability of the tool for a systems security engineer. Finally, we have also identified a number of areas for further research.

5.1 IDEA Program Conclusions

The IDEA proof-of-concept prototype has demonstrated that multiple risk assessment tools with different modes of operation can be combined to provide a more complete picture of a node's level of exposure. It has also demonstrated, through the use of OFE, that it is possible to identify nodes, across a network, which are at a greater level of risk.

The primary advantage of the IDEA prototype is its XML interface. Coupled with the promising research of NLP, the ability to expand support of scan tool outputs becomes a relatively simple operation.

The FuzzyFusion™ component was designed to be used in an enterprise distributed vulnerability assessment solution. Research into Levels of FuzzyFusion™ showed that the component fully supports distributed vulnerability detection. One of the current problems with assessing vulnerabilities across large networks is the amount of time necessary for a centralized server to scan all nodes. As a distributed component, FuzzyFusion™ provides vulnerability assessment in near real-time.

The FuzzyFusion™ component was scaled for the highest degree of reuse. The XML interface provides a standard communication mechanism. The small interface of the component provides enough capabilities to perform several vulnerability assessment tasks.

5.2 Areas for Improvement

One focus area for improving the usability and function of the current FuzzyFusion™ component is allowing the component to maintain node history and use historical information in the assessment of the node's Terminal Condition.

Implementing the FuzzyFusion™ End Condition calculations as described in Method 3 would bring the component up to date with the accumulation of research performed on IDEA.

Because of the nature of a component, the primary areas for improving the FuzzyFusion™ component reside in its use as a black box rather than in modifying its behavior or expanding its functionality. For example, it is quite possible to maintain history information, or perform alternate calculations for End Conditions outside of the FuzzyFusion™ component.

Along this line of approach for improvement, implementing NLP in mapping external vulnerability scanner tool results to FuzzyFusion™-internal database information would support quicker and possibly more accurate tool integration.

6 Future Work

As with any new research area, there are several directions and areas that could be explored for future research. The areas described below are topics that could be addressed in the next 18-36 months.

6.1 Real-time Vulnerability Analysis

One area of great interest is the capability to detect vulnerabilities in near real-time. To do so, a dynamic framework of vulnerabilities would have to be created. This would be the domain of existing scanning technology such as the STAT™ scanner or ISS Scanner. Once the vulnerability tools are in place, Fuzzy Fusion™ could be applied to correlate the results of multiple scans. With this approach, the end user would benefit from having diverse scanning technologies available and in use, with the additional assurance that a detected vulnerability exists with a high degree of confidence.

6.2 Real-time Intrusion Detection

Another area that could benefit from Fuzzy Fusion™ correlation is the area of real-time intrusion detection. Again, the correlation engine would be applicable to fuse the results of multiple intrusion detection sensors. With current generation intrusion detection technology, these outputs would be the results of comprehensive, attack signature or pattern based intrusion detection systems. Some research to date has indicated that less complex intrusion detection systems, consisting of more specialized sensors deployed at various points in the network, may be a more efficient intrusion or anomaly detection technology.

In this scenario, Fuzzy Fusion™ could be applied to correlate the results of the various sensor technologies into a cohesive end result. For example, if the specialized intrusion detection sensors detect different anomalies that would indicate an intrusion in progress, the correlation and resolution of the results would be generated by Fuzzy Fusion.™

6.3 Fixing Vulnerabilities

One area discussed in our final Blue Ribbon panel meeting was the capability to prioritize vulnerabilities for repair, based on a user defined priority scheme. For example, a rule of thumb in information assurance states that a user can have two out of three characteristics in his system: mission functionality, performance, or information assurance. In this scenario, Fuzzy Fusion™ could be used to optimize the equation and incorporate a cost metric to recommend which vulnerability should be addressed first when fixing a corporate network.

"If the company's CIO has \$100,000 to spend on fixing network vulnerabilities, how best can the money be spent?" Dr. Blaine Burnham.

The FuzzyFusion™ component can be made fully capable of addressing this issue by modifying it to trace the end conditions of the node back to the set of vulnerabilities which led to that set of end conditions.

Within the FuzzyFusion™ component, the database maintains the list of client inputs. These inputs contain vulnerability and OFE information (as well as trust and tool). To identify which vulnerability to address first, we take the highest end condition (System over User over Data) and trace it backwards to the intermediate condition(s) whose fuzzy membership rules lead to the end condition. We then take each of these intermediate

conditions and trace them back to their mapped IVEs and then back to the vulnerabilities mapping to the IVEs. If we sort these resulting vulnerabilities according to the value of their OFE information, we have an ordered list of vulnerabilities to address. Those vulnerabilities with the highest OFE values are fixed first, those with the next highest are then fixed, etc.

In response to the question raised by the Blue Ribbon panel it is relatively simple to produce a vulnerability fix-list for each node on a network. This fix-list could also take into account level 3 and level 4 exploitation path information which would direct the fix-list to individual nodes on the network.

Figure 6.3-1 shows, in red, a possible reverse mapping from the node condition, System End, to the vulnerabilities which led to it.

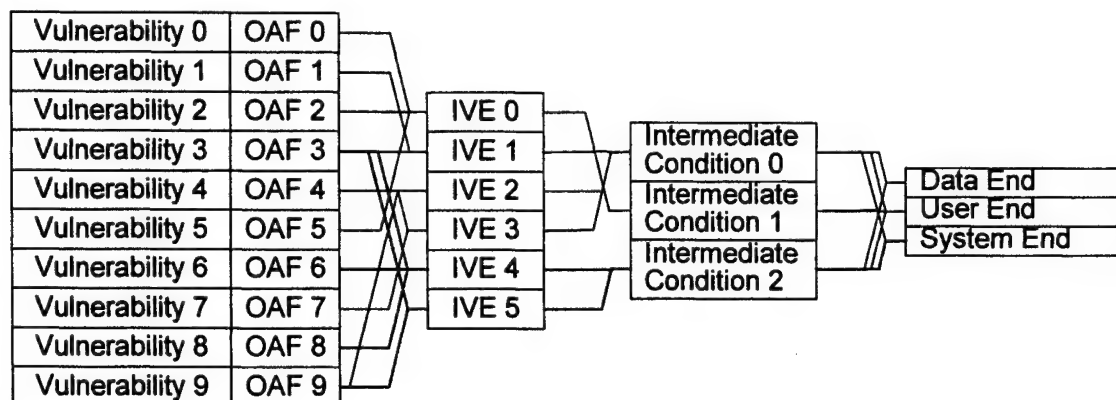


Figure 6.3-1– Mapping of the End conditions to the Vulnerabilities that led to it

6.4 Maintaining Histories

6.4.1 Trust

Can we modify the Trust modifier for a tool/vulnerability datum based on historical information? That is, can we generate a confidence index for a given tool's results based on the captured history of use information. For example, if, among several scanning tools, one historically misses significant vulnerabilities in the systems scanned, this tool would have a lower confident rating than other scanning tools.

6.5 Security Policy Meta-Language

One topic researched during IDEA was the concept of a security policy meta language. We developed the concept as a technique for policy specification that could make the policy specification process more meaningful to the end user. To do so, we applied knowledge engineering and extraction techniques to experienced security engineers to determine the appropriate syntax for a meta language. Then, we defined a set of possible noun and verb constructs to specify policy actions.

The meta-language, if developed as a separate concept, could have applicability to other areas of policy definition. For example, red teams characteristically express their actions in terms of goals and objectives. It would be possible to apply the meta-language to define attack-defend activities and express the results in a standard syntax. This would accommodate result correlation and allow better communication with the end user.

6.6 Generalization of Fuzzy Fusion™

Throughout this phase of our research, we have considered development of a more general-purpose Fuzzy Fusion™ engine that could be applied to correlation of multiple data types in various problem domains. On the advice of our Blue Ribbon panel, we suspended this line of research. The belief of our panelists was that there are already several general purpose tools in place in the research community. What is unique about Fuzzy Fusion™ is the domain specific application and its capability to solve multiple problems within this domain. It is their recommendation that we continue to evolve the correlation capabilities and generate a more comprehensive worked example in the information assurance domain, then expand outward to other domains.

6.7 Natural Language Processing of Vulnerabilities

Towards the end of our research, we considered the ability to automatically generate mappings between tool-produced vulnerabilities and the IDEA Taxonomy information. The capability to automatically generate these mappings would facilitate plugging in new vulnerability scan tools to the FuzzyFusion™ component.

Currently within the STAT® Analyzer database, there are over 2000 tool-generated vulnerabilities mapped. NLP promises automatic correlation between tool vulnerabilities and the IDEA Taxonomy.

7 Appendix A – Data

7.1 IVE Table

This table shows the IVE identifier and its meaning.

Vulnerability results from external tools, STAT in this exercise, are mapped to the

Table 7.1-1 – IVE information

Taxonomy Identifier	Description
Identification and Authentication Items	
IVE-001	No access without I+A
IVE-002	No user access without I+A
IVE-003	No user access from console without I+A
IVE-004	No user access from network without I+A
IVE-005	No root access without I+A
IVE-006	No root access from console without I+A
IVE-007	No root access from network without I+A
IVE-008	No privilege elevation without I+A
IVE-009	No privilege elevation from console without I+A
IVE-010	No privilege elevation from network without I+A
IVE-011	No user access from administrator
IVE-012	Users ID before access (individual or group)
IVE-013	Uniquely ID individuals only before access (no groups)
IVE-015	No guest accounts
IVE-016	No anonymous accounts
IVE-017	Do not ID via host
IVE-018	Do not ID host acting on behalf of user or group instead of user or group itself (.rhost)
IVE-019	ID host acting on behalf of user or group instead of user or group itself, as long as membership in group implies list of specific users in that group
IVE-020	No default accounts
IVE-021	Do not ID on remote component (IA server)
IVE-022	All users require authentication

Taxonomy Identifier	Description
IVE-023	No accounts without password
IVE-024	No user accounts without password
IVE-025	No domain user without password
IVE-026	No NULL Sessions
IVE-027	No guest accounts without password
IVE-028	No blank passwords present, but allowed
IVE-029	No root accounts without password
IVE-030	No domain root without password
IVE-031	Assign conditions requiring re-authentication
IVE-032	Protect/maintain authentication data
IVE-033	Stored passwords are encrypted
IVE-034	Password encryption strength meets policy
IVE-035	Encrypted passwords are protected (Access Control)
IVE-036	Passwords are not cached for user
IVE-037	Encrypted passwords are not accessible unencrypted
IVE-038	Protect previous (plaintext) passwords
IVE-039	Protect previous (ciphertext) passwords
IVE-040	Protect passwords in dump (core) files
IVE-041	Strong encryption method for passwords
IVE-042	Stored root passwords are encrypted
IVE-043	Root password encryption strength meets policy
IVE-044	Encrypted root passwords are protected (Access Control)
IVE-045	Protect user passwords from brute-force attacks
IVE-046	Maintain user password length suitable to data protected
IVE-047	Maintain user password life suitable to data protected
IVE-048	Enforce user password content (hard to guess)
IVE-049	System user generated passwords
IVE-050	Maintain user password history
IVE-051	Protect root passwords from brute-force attacks
IVE-052	Maintain root password length suitable to data protected

Taxonomy Identifier	Description
IVE-053	Maintain root password life suitable to data protected
IVE-054	Enforce root password content (hard to guess)
IVE-055	System generated root passwords
IVE-056	Maintain root password history
IVE-057	Authentication failures user
IVE-058	Maximum number of authentication failures user
IVE-059	Action taken on exceeding maximum auth failures user
IVE-060	Authentication failures actively monitored user
IVE-061	Authentication failures root
IVE-062	Maximum number of authentication failures root
IVE-063	Action taken on exceeding maximum auth failures root
IVE-064	Authentication failures actively monitored root
IVE-065	Protected (encrypt) authentication info passed b/w components
IVE-066	No yppasswd
IVE-067	No rlogin
IVE-068	No telnet
IVE-069	Encryption for network tx passwords meets policy
IVE-070	All authentication on network is encrypted
IVE-071	Unencrypted passwords have strong Access Control
IVE-072	Unencrypted stored user passwords have strong access control
IVE-073	Unencrypted stored root passwords have strong access control
IVE-085	Protected mechanism for authentication
IVE-086	Mechanism resistant to spoofing
IVE-087	No user credential transmit without notice
IVE-088	Ensure single-use authentication employs single-use authentication data.
IVE-089	Multiple authentication mechanisms assigned to events
IVE-090	Account Servers admin console login only
IVE-091	Protect Accounts
IVE-092	Protect user accounts
IVE-093	No stale user accounts

Taxonomy Identifier	Description
IVE-094	Maintain user login hours
IVE-095	Protect User Information
IVE-096	Protect against user name enumeration
IVE-097	Protect root accounts
IVE-098	No stale root accounts
IVE-099	Maintain root login hours
IVE-100	Protect Root Information
IVE-101	No information utilities
IVE-102	No Finger
IVE-103	No Rusers
IVE-104	No Rwho
IVE-105	No Whois
IVE-106	Specify Login Environment
IVE-107	Specify data provided during I+A
IVE-108	TSF can specify set of actions user can take prior to identification
IVE-109	Maintain authorization and clearance data as well as authentication data for users
IVE-110	Associate authorization and clearance data with users
Access Control	
IVE-111	Must have DAC
IVE-112	DAC with specified users/groups
IVE-113	DAC enforced
IVE-114	Must have MAC
IVE-115	Account permissions
IVE-116	Guest Account permissions meet policy
IVE-117	Guest can login only at console
IVE-118	Domain Guest Account permissions meet policy
IVE-119	Anonymous Account permissions meet policy
IVE-120	Domain Anonymous Account permissions meet policy

Taxonomy Identifier	Description
IVE-121	User Account permissions meet policy
IVE-122	Domain User Account permissions meet policy
IVE-123	Root Account permissions meet policy
IVE-124	Domain Root Count permissions meet policy
IVE-125	Disk Access is fully controlled
IVE-126	Network File System (NFS) use meets policy
IVE-127	NFS is not permitted
IVE-128	NFS has access control
IVE-129	NFS resistant to spoofing
IVE-130	NFS does not export system files
IVE-131	TFS is not permitted
IVE-132	NetBIOS Disk Share use meets policy
IVE-133	NetBIOS shares are not permitted
IVE-134	NetBIOS shares have security
IVE-135	NetBIOS shares have Access Control
IVE-136	NetBIOS shares have write protection
IVE-137	File System meets policy
IVE-138	No FAT
IVE-139	Don't provide file listings
IVE-140	Don't provide directory information
IVE-141	Don't provide directory structure information
IVE-142	Peripherals restricted to console user
IVE-143	Only console user can access CD-ROM
IVE-144	Only console user can access floppy
IVE-145	Must have access control for file reads
IVE-146	External sources cannot bypass local control for file reads
IVE-147	No external file reads via HTML
IVE-148	No external file reads via vb macros (outlook, word, etc)
IVE-149	No external file reads via cgi
IVE-150	No external file reads via applets

Taxonomy Identifier	Description
IVE-151	No external file reads via activeX
IVE-152	No external file reads via cookies
IVE-153	No external file reads via Java
IVE-154	No external file reads via Active Server Page (.asp)
IVE-155	No external file reads via mail uuencode
IVE-156	No external file reads via WWW access
IVE-157	No external file reads via FTP
IVE-158	No external file reads via TFTP
IVE-159	No external file reads via SNMP
IVE-160	No outside control over file read as root
IVE-161	Enforce access control for local reading files
IVE-162	Local user cannot bypass local control for file reads
IVE-163	Local user cannot bypass traverse checking for file reads
IVE-164	Must have access control for file writes
IVE-165	Cannot write executable file with potential execution
IVE-166	External sources cannot bypass local control for file writes
IVE-167	No external file writes via HTML
IVE-168	No external file writes via vb macros (outlook, word, etc)
IVE-169	No external file writes via cgi
IVE-170	No external file writes via applets
IVE-171	No external file writes via activeX write
IVE-172	No external file writes via cookies
IVE-173	No external file writes via Java
IVE-174	No external file writes via Active Server Page (.asp)
IVE-175	No external file writes via mail uuencode
IVE-176	No external file writes via WWW write access
IVE-177	No external file writes via FTP write
IVE-178	No external file writes via TFTP
IVE-179	No external file writes via SNMP
IVE-180	No outside control over file write as root

Taxonomy Identifier	Description
IVE-181	Cannot write potentially malicious executables
IVE-182	Enforce access control for local writing files
IVE-183	Local user cannot bypass local control for file writes
IVE-184	Local user cannot bypass traverse checking for file writes
IVE-185	Access Control over system files
IVE-186	System files controlled by root
IVE-187	Root only can read of system files
IVE-188	Root only can write of system files
IVE-189	External sources cannot bypass local control for system file access
IVE-190	Domain Root read of system files only
IVE-191	Domain Root write of system files only
IVE-192	Domain Root read of Domain system files
IVE-193	Domain Root write of Domain system files
IVE-194	Protect system files from external access
IVE-195	No system files on shared drives
IVE-196	Must have access control over executable files
IVE-197	External sources cannot bypass local control for file execution
IVE-198	No external file execution via HTML
IVE-199	No external file execution via vb macros (outlook, word, etc)
IVE-200	No external file execution via cgi
IVE-201	No external file execution via applets
IVE-202	No external file execution via activeX
IVE-203	No external file execution via cookies
IVE-204	No external file execution via Java
IVE-205	No external file execution via Active Server Page (.asp)
IVE-206	No external file execution via mail
IVE-207	No external file execution via WWW (download active content)
IVE-208	No external file execution via FTP
IVE-209	No external file execution via rcmd
IVE-210	No external file execution via rsh

Taxonomy Identifier	Description
IVE-211	No external file execution via rpc
IVE-212	No external execution via X windows
IVE-213	No external execution via font install
IVE-214	No external execution via IFRAMEs
IVE-215	Control .exe install
IVE-216	Control .exe install from media
IVE-217	Control .exe install from network
IVE-218	Must have access control over execution as root
IVE-219	No CD autorun
IVE-220	No outside control over file execution as root
IVE-221	Warn before executing program from external source
IVE-222	Warn before executing at VB scripts from MS Office product
IVE-223	Access Control over resources
IVE-224	No external control over resources
IVE-225	Object Permissions
IVE-226	Must have access control over DCOM use
IVE-227	DCOM can be accessed by administrator only
IVE-228	DCOM can be accessed by administrator at console only
IVE-229	Must have access control over CORBA use
IVE-230	Object Reuse
IVE-231	Protect deleted files
IVE-232	Protect passwords
IVE-233	Protect Clipboard
IVE-234	Access Control enforced for applications
IVE-235	AC for Databases
IVE-236	AC for internal dB read
IVE-237	AC for remote dB read
IVE-238	Protect Network Access
IVE-239	Protect Network Access via Router
IVE-241	protect router config files

Taxonomy Identifier	Description
IVE-242	protect against inadvertent use of multihome devices as routers
Communication Protection	
IVE-243	Protect Network Access via Firewall
IVE-244	Protect Firewall
IVE-245	Protect Firewall Configuration
IVE-246	Protect Network Access via Proxy Firewall
IVE-247	Protect Network Access via Application Firewall
IVE-248	Protect network access via modem
IVE-249	Protect from data disclosure over the network
IVE-250	Encryption
IVE-251	All data sent to network encrypted
IVE-252	Generate encryption to meet metric
IVE-253	Protect keys
IVE-254	Protect Data exchange with web saves
IVE-255	Certificates for secure sites
IVE-256	All data sent to secure site encrypted
IVE-257	Protect Access to the data stream
IVE-258	Restrict services that may disclose data
IVE-259	UUCP
IVE-260	Gopher
IVE-261	Ensure Authenticity
IVE-262	Detect/Prevent forged/copied data (spoofing)
IVE-263	Ensure data exchange established with addressed peer entity
IVE-264	Ensure data source is the one claimed
IVE-265	Protect against port hijacking
IVE-266	Enable NT TCP/IP security (restrict port use, protocols, etc)
IVE-267	Protect port use
IVE-268	Ensure data integrity

Taxonomy Identifier	Description
Node Protection	
IVE-269	Protect Availability
IVE-270	Protect against Denial of Service
IVE-271	No DOS programs present
IVE-272	Not vulnerable to DOS programs
IVE-273	No Distributed Denial of Service programs present
IVE-274	Protect against performance degradation
IVE-275	No DOS vulnerabilities
IVE-276	Protect against malicious software
IVE-277	No Trojan horse
IVE-278	No Trojan horses on system
IVE-279	Protect against Trojan Horses
IVE-280	Backdoor
IVE-281	No backdoors on system
IVE-282	Protect against backdoors
IVE-283	No potentially compromised remote control utilities
IVE-284	Virus
IVE-285	Protect against viruses
IVE-286	No viruses present
IVE-287	Protect against worms
IVE-288	No worms present
IVE-289	No potentially malicious software
IVE-290	No password crackers
IVE-291	No keyboard sniffers
IVE-292	No port scanners
IVE-293	No info gatherers
IVE-294	No Privilege Elevation Software
IVE-295	No Buffer Overflow
IVE-296	Detect compromised system
IVE-297	Suspicious Registry settings

Taxonomy Identifier	Description
Operational Procedure	
IVE-298	Controlled Operational Procedures
IVE-299	Startup/Shutdown procedure
IVE-300	Data Labeling
IVE-301	ID non-encrypted data sent to web
IVE-302	ID entering secure site
IVE-303	ID mismatched web certificates
IVE-304	ID secure data in browser
IVE-305	ID web redirects
IVE-306	Do not allow data from secure web page to be stored on local hard drive
IVE-307	Backup Procedure
IVE-308	General
IVE-309	No Out Of Date (OOD) OS
IVE-310	No OOD Software
IVE-311	No OOD Software allowing privilege elevation
IVE-312	No OOD Software allowing execution of a program as root
IVE-313	No OOD Software allowing DOS
IVE-314	System Info
IVE-315	No Unreliable SW
IVE-316	No POSIX on NT
IVE-317	No OS/2 on NT
IVE-318	Legal
IVE-319	Intrusion Detection
IVE-320	Port Scan
IVE-321	Enumeration (other then user name)
Auditing	
IVE-322	Perform Auditing
IVE-323	Perform System wide auditing

Taxonomy Identifier	Description
IVE-324	Associate auditable actions with identity
IVE-325	Control creation of Audit events
IVE-326	Protect audit files
IVE-327	Restrict access to audit files
IVE-328	Restrict creation of audit events
IVE-329	Don't overwrite audit files
IVE-330	Provide alternate storage for full audit partition
IVE-331	Shut down when Audit full
IVE-332	Do not shut down when Audit is full
IVE-333	Provide sufficient storage for log files
IVE-334	Provide sufficient duration for log files to be saved
IVE-335	Protect auditing mechanism
IVE-336	Audit required events
IVE-337	Audit processes
IVE-338	Audit security processes
IVE-339	Audit system events
IVE-340	Audit backup events
IVE-341	Audit user management
IVE-342	Audit logon/off activity
IVE-343	Audit file/object access
IVE-344	Audit system file/object access
IVE-345	Audit Comms Accesses
IVE-346	Audit Anonymous Logins
IVE-347	Audit HTTP requests
IVE-348	Audit ftp events
IVE-349	Audit active directory events
IVE-350	Audit remote file access (NFS, NetBIOS)
IVE-351	Audit remote access (RAS, etc)
IVE-352	Audit Java
IVE-353	Maintain memory dumps

7.2 Condition Table

Table 7.2-1 – Vulnerability Categories and their definitions

Vulnerability Category	Definition
Account Access	Access to account information: Passwords, privileges, logon names, etc. for accounts.
User Access	Access as a non-administrative user of the system.
Administrator Access	Access to administrative privileges.
Privilege Access	Access to portions of the OS that allows privilege elevation.
Password Access	Access to files containing passwords – plaintext or ciphertext.
Network Access	Access to network information, machines, or ports.
File Access	Access to the filesystem.
Data Access	Access to data being delivered across a network.
Hijack Access	Access to applications that allow the external user to take control of the system.
Backdoor Access	Access to the system through backdoor applications.
Information Access	Access to hardware or software information (port information, OS types, etc.)
Hardware Access	Access to devices on the machine (i.e. CD-Rs, Memory, BIOS, etc.)
Encryption	Access to encrypted information.
Process Access	Access to system or user processes.
Compromised	System is already compromised.
TBI	Mapping of a vulnerability that is currently un-catalogued.

7.3 Rule Table

IAT-ID	Pre Condition	Post Condition
IVE-001	Start	System
IVE-002	Start	Account
IVE-003	Start	Account
IVE-004	Start	Account

IVE-005	Start	Administrator
IVE-006	Start	Administrator
IVE-007	Start	Administrator
IVE-008	Start	Privilege
IVE-009	Start	Privilege
IVE-010	Start	Privilege
IVE-011	Start	Administrator
IVE-012	Start	Account
IVE-013	Start	Account
IVE-014	Start	TBI
IVE-015	Start	Account
IVE-016	Start	Account
IVE-017	Start	Account
IVE-018	Start	Account
IVE-019	Start	Account
IVE-020	Start	Account
IVE-021	Start	Account
IVE-022	Start	Account
IVE-023	Start	Account
IVE-024	Start	Account
IVE-025	Start	Account
IVE-026	Start	Account
IVE-027	Start	Account
IVE-028	Start	Account
IVE-029	Start	Administrator
IVE-030	Start	Administrator
IVE-031	Start	Account
IVE-032	Start	Encryption
IVE-033	Start	Password
IVE-034	Start	Password
IVE-035	Start	Password
IVE-036	Start	Password
IVE-037	Start	Password
IVE-038	Start	Password

IVE-039	Start	Password
IVE-040	Start	Password
IVE-041	Start	Password
IVE-042	Start	Password
IVE-043	Start	Password
IVE-044	Start	Password
IVE-045	Start	Password
IVE-046	Start	Password
IVE-047	Start	Password
IVE-048	Start	Password
IVE-049	Start	Password
IVE-050	Start	Audit
IVE-051	Start	Password
IVE-052	Start	Password
IVE-053	Start	Password
IVE-054	Start	Password
IVE-055	Start	Password
IVE-056	Start	Password
IVE-057	Start	Account
IVE-058	Start	Account
IVE-059	Start	Account
IVE-060	Start	Account
IVE-061	Start	Account
IVE-062	Start	Account
IVE-063	Start	Account
IVE-064	Start	Account
IVE-065	Start	Data
IVE-066	Start	Account
IVE-067	Start	Account
IVE-068	Start	Account
IVE-069	Start	Password
IVE-070	Start	Password
IVE-071	Start	Password
IVE-072	Start	Password

IVE-073	Start	Password
IVE-074	Start	TBI
IVE-075	Start	TBI
IVE-076	Start	TBI
IVE-077	Start	TBI
IVE-078	Start	TBI
IVE-079	Start	TBI
IVE-080	Start	TBI
IVE-081	Start	TBI
IVE-082	Start	TBI
IVE-083	Start	TBI
IVE-084	Start	TBI
IVE-085	Start	Account
IVE-086	Start	Account
IVE-087	Start	Account
IVE-088	Start	Account
IVE-089	Start	Account
IVE-090	Start	Administrator
IVE-091	Start	Account
IVE-092	Start	Account
IVE-093	Start	Account
IVE-094	Start	Account
IVE-095	Start	Account
IVE-096	Start	Account
IVE-097	Start	Account
IVE-098	Start	Account
IVE-099	Start	Account
IVE-100	Start	Account
IVE-101	Start	Account
IVE-102	Start	Account
IVE-103	Start	Account
IVE-104	Start	Account
IVE-105	Start	Account
IVE-106	Start	Account

IVE-107	Start	Account
IVE-108	Start	Account
IVE-109	Start	Account
IVE-110	Start	Account
IVE-111	Start	Account
IVE-112	Start	Account
IVE-113	Start	Account
IVE-114	Start	Account
IVE-115	Start	Account
IVE-116	Start	Account
IVE-117	Start	Account
IVE-118	Start	Account
IVE-119	Start	Account
IVE-120	Start	Account
IVE-121	Start	Account
IVE-122	Start	Account
IVE-123	Start	Account
IVE-124	Start	Account
IVE-125	Start	Hardware
IVE-126	Start	Network
IVE-127	Start	Network
IVE-128	Start	Network
IVE-129	Start	Network
IVE-130	Start	Network
IVE-131	Start	Network
IVE-132	Start	Hardware
IVE-133	Start	Hardware
IVE-134	Start	Hardware
IVE-135	Start	Hardware
IVE-136	Start	Hardware
IVE-137	Start	File
IVE-138	Start	File
IVE-139	Start	File
IVE-140	Start	File

IVE-141	Start	File
IVE-142	Start	Hardware
IVE-143	Start	Hardware
IVE-144	Start	Hardware
IVE-145	Start	File
IVE-146	Start	File
IVE-147	Start	File
IVE-148	Start	File
IVE-149	Start	File
IVE-150	Start	File
IVE-151	Start	File
IVE-152	Start	File
IVE-153	Start	File
IVE-154	Start	File
IVE-155	Start	File
IVE-156	Start	File
IVE-157	Start	File
IVE-158	Start	File
IVE-159	Start	File
IVE-160	Start	File
IVE-161	Start	File
IVE-162	Start	File
IVE-163	Start	File
IVE-164	Start	File
IVE-165	Start	File
IVE-166	Start	File
IVE-167	Start	File
IVE-168	Start	File
IVE-169	Start	File
IVE-170	Start	File
IVE-171	Start	File
IVE-172	Start	File
IVE-173	Start	File
IVE-174	Start	File

IVE-175	Start	File
IVE-176	Start	File
IVE-177	Start	File
IVE-178	Start	File
IVE-179	Start	File
IVE-180	Start	File
IVE-181	Start	File
IVE-182	Start	File
IVE-183	Start	File
IVE-184	Start	File
IVE-185	Start	File
IVE-186	Start	File
IVE-187	Start	File
IVE-188	Start	File
IVE-189	Start	File
IVE-190	Start	File
IVE-191	Start	File
IVE-192	Start	File
IVE-193	Start	File
IVE-194	Start	File
IVE-195	Start	File
IVE-196	Start	Hijack
IVE-197	Start	Hijack
IVE-198	Start	Hijack
IVE-199	Start	Hijack
IVE-200	Start	Hijack
IVE-201	Start	Hijack
IVE-202	Start	Hijack
IVE-203	Start	Hijack
IVE-204	Start	Hijack
IVE-205	Start	Hijack
IVE-206	Start	Hijack
IVE-207	Start	Hijack
IVE-208	Start	Hijack

IVE-209	Start	Hijack
IVE-210	Start	Hijack
IVE-211	Start	Hijack
IVE-212	Start	Hijack
IVE-213	Start	Hijack
IVE-214	Start	Hijack
IVE-215	Start	Hijack
IVE-216	Start	Hijack
IVE-217	Start	Hijack
IVE-218	Start	Hijack
IVE-219	Start	Hijack
IVE-220	Start	Hijack
IVE-221	Start	Hijack
IVE-222	Start	Hijack
IVE-223	Start	TBI
IVE-224	Start	TBI
IVE-225	Start	Process
IVE-226	Start	Process
IVE-227	Start	Process
IVE-228	Start	Process
IVE-229	Start	Process
IVE-230	Start	Process
IVE-231	Start	File
IVE-232	Start	Password
IVE-233	Start	File
IVE-234	Start	File
IVE-235	Start	File
IVE-236	Start	File
IVE-237	Start	File
IVE-238	Start	Network
IVE-239	Start	Network
IVE-240	Start	TBI
IVE-241	Start	Hardware
IVE-242	Start	Hardware

IVE-243	Start	Network
IVE-244	Start	Network
IVE-245	Start	File
IVE-246	Start	Network
IVE-247	Start	Network
IVE-248	Start	Network
IVE-249	Start	Data
IVE-250	Start	Data
IVE-251	Start	Data
IVE-252	Start	Data
IVE-253	Start	Data
IVE-254	Start	Data
IVE-255	Start	Data
IVE-256	Start	Data
IVE-257	Start	Data
IVE-258	Start	Data
IVE-259	Start	Data
IVE-260	Start	Data
IVE-261	Start	Data
IVE-262	Start	Data
IVE-263	Start	Data
IVE-264	Start	Data
IVE-265	Start	Data
IVE-266	Start	Data
IVE-267	Start	Data
IVE-268	Start	Data
IVE-269	Start	Process
IVE-270	Start	Process
IVE-271	Start	Process
IVE-272	Start	Process
IVE-273	Start	Process
IVE-274	Start	Process
IVE-275	Start	Process
IVE-276	Start	Hijack

IVE-277	Start	Hijack
IVE-278	Start	Compromised
IVE-279	Start	Hijack
IVE-280	Start	Backdoor
IVE-281	Start	Backdoor
IVE-282	Start	Backdoor
IVE-283	Start	Hijack
IVE-284	Start	Hijack
IVE-285	Start	Hijack
IVE-286	Start	Compromised
IVE-287	Start	Hijack
IVE-288	Start	Compromised
IVE-289	Start	Compromised
IVE-290	Start	Compromised
IVE-291	Start	Compromised
IVE-292	Start	Compromised
IVE-293	Start	Compromised
IVE-294	Start	Compromised
IVE-295	Start	Process
IVE-296	Start	Compromised
IVE-297	Start	Compromised
IVE-298	Start	Account
IVE-299	Start	Account
IVE-300	Start	Data
IVE-301	Start	Data
IVE-302	Start	Data
IVE-303	Start	Data
IVE-304	Start	Data
IVE-305	Start	Data
IVE-306	Start	Data
IVE-307	Start	TBI
IVE-308	Start	TBI
IVE-309	Start	System
IVE-310	Start	System

IVE-311	Start	System
IVE-312	Start	System
IVE-313	Start	System
IVE-314	Start	System
IVE-315	Start	System
IVE-316	Start	System
IVE-317	Start	System
IVE-318	Start	TBI
IVE-319	Start	TBI
IVE-320	Start	TBI
IVE-321	Start	TBI
IVE-322	Start	Audit
IVE-323	Start	Audit
IVE-324	Start	Audit
IVE-325	Start	Audit
IVE-326	Start	Audit
IVE-327	Start	Audit
IVE-328	Start	Audit
IVE-329	Start	Audit
IVE-330	Start	Audit
IVE-331	Start	Audit
IVE-332	Start	Audit
IVE-333	Start	Audit
IVE-334	Start	Audit
IVE-335	Start	Audit
IVE-336	Start	Audit
IVE-337	Start	Audit
IVE-338	Start	Audit
IVE-339	Start	Audit
IVE-340	Start	Audit
IVE-341	Start	Audit
IVE-342	Start	Audit
IVE-343	Start	Audit
IVE-344	Start	Audit

IVE-345	Start	Audit
IVE-346	Start	Audit
IVE-347	Start	Audit
IVE-348	Start	Audit
IVE-349	Start	Audit
IVE-350	Start	Audit
IVE-351	Start	Audit
IVE-352	Start	Audit
IVE-353	Start	Audit

7.4 CVE Information

CVE
CVE-1999-0012
CVE-1999-0016
CVE-1999-0077
CVE-1999-0079
CVE-1999-0103
CVE-1999-0103
CVE-1999-0103
CVE-1999-0116
CVE-1999-0116
CVE-1999-0128
CVE-1999-0128
CVE-1999-0153
CVE-1999-0153
CVE-1999-0177
CVE-1999-0177
CVE-1999-0178
CVE-1999-0191
CVE-1999-0224
CVE-1999-0224

CVE-1999-0225
CVE-1999-0225
CVE-1999-0227
CVE-1999-0227
CVE-1999-0228
CVE-1999-0228
CVE-1999-0233
CVE-1999-0233
CVE-1999-0233
CVE-1999-0265
CVE-1999-0265
CVE-1999-0274
CVE-1999-0274
CVE-1999-0275
CVE-1999-0275
CVE-1999-0278
CVE-1999-0281
CVE-1999-0288
CVE-1999-0292
CVE-1999-0292
CVE-1999-0344
CVE-1999-0344
CVE-1999-0348
CVE-1999-0349
CVE-1999-0366
CVE-1999-0372
CVE-1999-0376
CVE-1999-0376
CVE-1999-0382
CVE-1999-0384
CVE-1999-0385
CVE-1999-0407
CVE-1999-0449
CVE-1999-0458

CVE-1999-0468
CVE-1999-0496
CVE-1999-0496
CVE-1999-0513
CVE-1999-0513
CVE-1999-0669
CVE-1999-0700
CVE-1999-0701
CVE-1999-0701
CVE-1999-0715
CVE-1999-0716
CVE-1999-0717
CVE-1999-0721
CVE-1999-0723
CVE-1999-0725
CVE-1999-0725
CVE-1999-0726
CVE-1999-0728
CVE-1999-0736
CVE-1999-0737
CVE-1999-0738
CVE-1999-0739
CVE-1999-0755
CVE-1999-0766
CVE-1999-0777
CVE-1999-0790
CVE-1999-0793
CVE-1999-0794
CVE-1999-0802
CVE-1999-0839
CVE-1999-0858
CVE-1999-0861
CVE-1999-0867
CVE-1999-0869

CVE-1999-0870
CVE-1999-0871
CVE-1999-0871
CVE-1999-0874
CVE-1999-0876
CVE-1999-0876
CVE-1999-0877
CVE-1999-0886
CVE-1999-0891
CVE-1999-0898
CVE-1999-0898
CVE-1999-0899
CVE-1999-0909
CVE-1999-0917
CVE-1999-0951
CVE-1999-0969
CVE-1999-0969
CVE-1999-0980
CVE-1999-0981
CVE-1999-0994
CVE-1999-0995
CVE-1999-0999
CVE-1999-1918
CVE-2000-0025
CVE-2000-0089
CVE-2000-0097
CVE-2000-0098
CVE-2000-0121
CVE-2000-0156
CVE-2000-0162
CVE-2000-0200
CVE-2000-0202
CVE-2000-0211
CVE-2000-0232

CVE-2000-0246
CVE-2000-0304
CVE-2000-0304
CVE-2000-0305
CVE-2000-0323
CVE-2000-0327
CVE-2000-0328
CVE-2000-0329
CVE-2000-0331
CVE-2000-0377
CVE-2000-0403
CVE-2000-0404
CVE-2000-0408
CVE-2000-0419
CVE-2000-0464
CVE-2000-0464
CVE-2000-0567
CVE-2000-0597
CVE-2000-0597
CVE-2000-0603
CVE-2000-0630
CVE-2000-0630
CVE-2000-0631
CVE-2000-0637
CVE-2000-0654
CVE-2000-0654
CVE-2000-0673
CVE-2000-0768
CVE-2000-0886
CVE-2000-1147
CVE-2001-0006
CVE-2001-0016
CVE-2001-0018
CVE-2001-0047

CVE-2001-0083
CVE-2001-0092
CVE-2001-0096
CVE-2001-0096
CVE-2001-0145

7.5 Fuzzy Set Contributions

This table is the fuzzy membership functions for the three fuzzy sets: Data_End, User_End, and System_End.

Table 7.5-1 – Fuzzy Membership Functions

IDEA Vulnerability Categories	Contributions
Account Access	.3 - User .7 - System
Administrator Access	1.0 - System
Backdoor Access	.4 - User .6 - System
Data Access	.3 - Data .7 - User
File Access	1.0 - User
Hardware Access	1.0 - Data
Hijack Access	.6 - User .4 - System
Information Access	1.0 - Data
Network Access	.4 - Data .3 - User .3 - System
Password Access	1.0 - System
Privilege Access	1.0 - System
Process Access	.3 - User .7 - System

User Access	1.0 – User
Compromised	1.0 – System
Encryption	.4 – Data .2 – User .4 – System
TBI	.3 – Data .3 – User .4 – System

8 Appendix B – XML Information

This appendix provides background information on XML as well as a rationale as to its selection as an interface-protocol for the FuzzyFusion™ component.

8.1 XML – A Better Component Protocol

Modern software applications are increasingly being built by assembling components. Components are typically binary units that provide services, and are often developed in isolation from the applications they will eventually run within. Components speak and are spoken to only through their interfaces, which are specified by an Interface Definition Language, or IDL. The final piece of the puzzle is the ORB (Object Request Broker), where a client goes to enlist the services of a component. The ORB enables distribution and location transparency (in other words, the client doesn't need to know where the component is running) by using special objects called proxies and stubs, which are responsible for intercepting and forwarding messages intended for components and their clients. The ORB is typically provided as part of a larger COTS (Commercial-Off-The-Shelf) package, while the proxies and stubs are typically generated from the IDL. These three parts form the part of the architecture called the *middleware*.

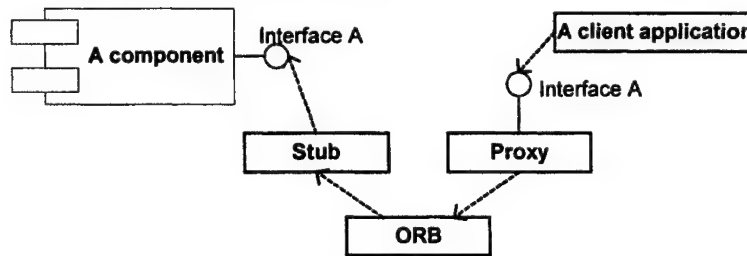


Figure 8.1-1 – Basic middleware architecture

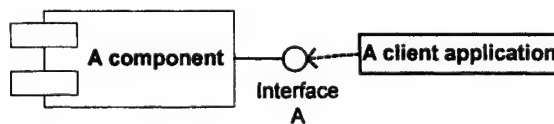


Figure 8.1-2 – A simplified view

This has proven to be a very flexible and powerful architecture over the past several years, and is essentially the same in COM/COM+, CORBA, and Enterprise JavaBeans, the leading component models. Benefits of component-based architectures include:

- Components, while sometimes difficult to develop, are relatively easy to integrate into applications
- Interfaces allow clients and components to upgrade concurrently and (usually) gracefully
- Components can be developed in virtually any programming language (with an exception being JavaBeans)

- Components provide location transparency
- Components support services such as security, transaction management and resource pooling

Agent technology adds other benefits, such as the ability for a component-agent to change the machine where it is physically executing at run time. In summary, components enable scalable, robust and flexible applications. However, the technology does have its limitations. First, many organizations have components developed for multiple component models, and these different models are not immediately compatible. This problem has been partially addressed by "bridges" between dissimilar ORBs.

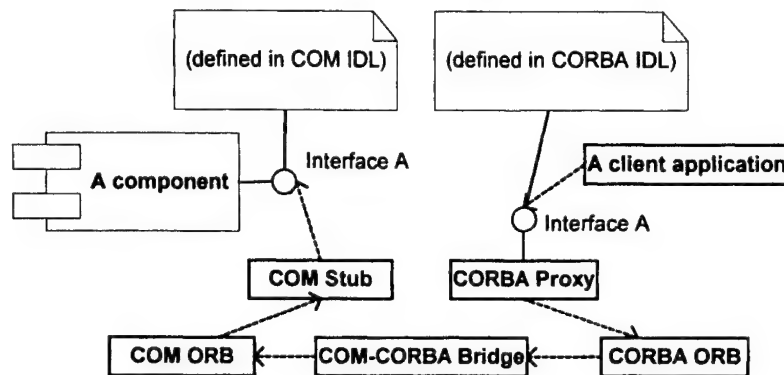


Figure 8.1-3 – Bridge between two dissimilar component models

Secondly, this architecture requires that an ORB is running on every machine where there is a client or server. This is especially an issue with web-enabled systems and nonstandard user interfaces such as PDAs. Initially it was thought that the solution was to add the ORB to the browser, as Java and ActiveX (COM under a different name) have done. However, security concerns have led many organizations to disable support for running these components in their browsers. Web-based GUIs tend to favor the simpler HTTP (HyperText Transfer Protocol) over the heavier-weight ORB protocols, because of HTTP's near-universal support, browser independence, and ease passing through server firewalls. Before that last bit scares you too much, recall that HTTP also has a very mature set of security frameworks built for it (such as secure connections and client authentication). So for a client that can only use HTTP, how does it access the services of a component?

Finally, assuming all these obstacles are overcome, what happens if the interfaces don't exactly agree? This is a very real situation caused by client and server version clashes, and components provided by different vendors. One solution is to put an intermediary, or adapter, between the client and component, but that solution still is limited in that it converts one exact interface to another exact interface. The adapter needs as many flavors as there are combinations of interfaces it has to adapt.

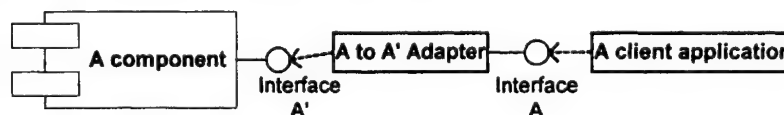


Figure 8.1-4 – Interface conflict with adapter solution

8.2 XML

Many solutions to these problems are provided by XML. XML, the Extensible Markup Language, is a universal syntax for describing and structuring data independent from the application logic. XML can be used to define unlimited languages for specific industries and applications. XML promises to simplify and lower the cost of data interchange and publishing in a Web environment. XML is a text-based syntax that is readable by both computer and humans. XML offers data portability and reusability across different platforms and devices. It is also flexible and extensible, allowing new tags to be added without breaking an existing document structure. XML was originally developed to provide more flexibility in formatting and displaying information on web pages. It is similar to HTML in that it is text-based, and uses tags to "mark up" text. The key difference is that HTML has fixed tags, such as for bold text, while XML allows user-defined tags.

```

<?xml version="1.0" ?>
<!DOCTYPE PERSON SYSTEM "person.dtd">
<PERSON AGE="19" GENDER="female" HAIR_COLOR="brown">
    <NAME>
        <FIRST>Jane</FIRST>
        <LAST>Doe</LAST>
    </NAME>
    <ADDRESS>
        <STREET1>123 Mockingbird Lane</STREET>
        <CITY>Anytown</CITY>
        <STATE>New Jersey</STATE>
    </ADDRESS>
</PERSON>

```

Figure 8.2-1 – An XML Document

In the above document, the tags PERSON, AGE, GENDER, NAME, etc. are all user-defined. The defined tags for an XML document define a *vocabulary*, which in this example is contained in a separate document called "person.dtd". DTD stands for "Document Type Definition". Because the tags (i.e. <STATE>) are included as text with the data (i.e. New Jersey), this is known as *self-describing data*. In other words, it is easier for a person or a machine to interpret what the contents mean.

Because of the simplicity and flexibility of XML, as well as the fact that it is an open standard, it has gained massive popularity, and has been appropriated for uses far beyond its original intent. One of these uses is messaging between components and clients. This is the approach taken by IDEA. In this approach, a client and a component both speak XML, and while XML is technically the interface between the two, the real interface is the vocabulary the two agree on.

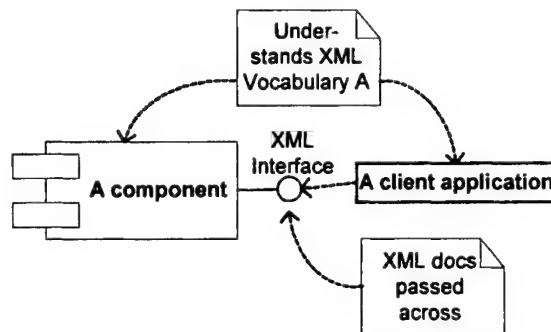


Figure 8.2-2 – XML Interface

8.2.1.1 XML as component interface

Unlike typical component interfaces, which are specific to components as well as to the component model, an XML interface is very simple - it doesn't know about any application-specific vocabulary. Because of this, the software architect can implement this interface in many ways, as appropriate to the component models (or lack thereof). Let's examine some of the challenges posed by components and how XML addresses them.

8.2.2 Dissimilar Component Models

Recall that in the previously discussed solution, a *bridge* would sit between the two ORBs and translate. Because of the complexity of inter-ORB communication, we would seldom want to write a bridge; rather we would look for a COTS product. Assuming we could find one, we now have three COTS products that all have to be kept current and compatible. This is no trivial issue, especially when you consider the features being added and modified by middleware all the time (such as asynchronous messaging, transaction support, etc.). In many cases we'll still end up writing "glue code" to bridge the gaps.

Bridging different component models can be easily done by creating a simple XML bridge. Why is this bridge any simpler than the bridge between ORBs? Since the format of XML documents is independent of application messages, a bridge can be created which simply passes an XML document from one component model to another. In addition, as the bridge is not COTS and is likely not using any ORB-specific functionality, it will usually not be necessary to create a new bridge when the ORB versions change. Is this bridge glue code? Sure, but it's not very much code, and often it's better to write a little glue code than to hardwire COTS products together and put yourself at their mercy.

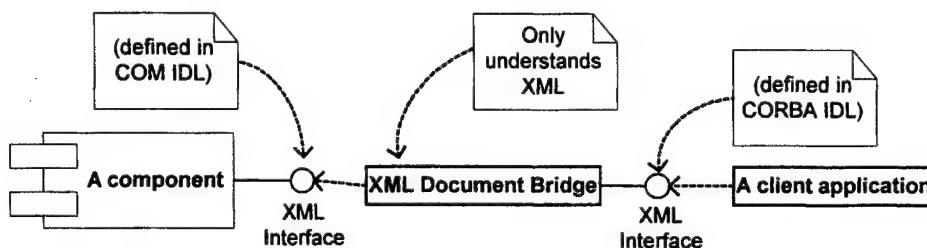


Figure 8.2-3 – XML Bridge

8.2.2.1 XML as bridge between component models

8.2.3 Lack of ORB Support

Similarly, XML removes the need for an ORB to be running on every platform. Since XML was developed as a web technology, it is recognized by HTTP already, and web development tools are well equipped to handle it. Here is an example architecture using a servlet to pass XML to and from a component:

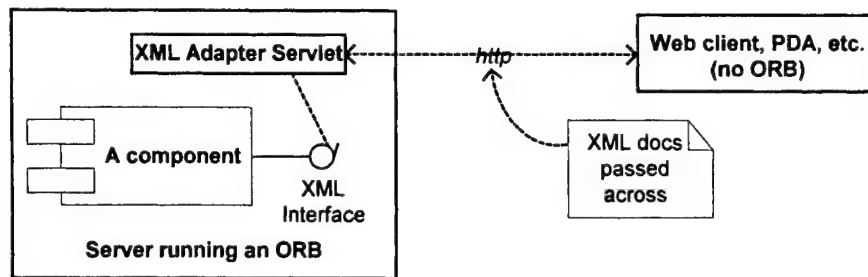


Figure 8.2-4 – XML Adapter

8.2.3.1 XML from web client to ORB-hosted component

In the case where a component is packaged as a mobile agent (such as certain proposed deployments of IDEA), the use of XML to talk to the agent is a natural fit. Simplifying the interface by using XML reduces the amount of code necessary to handle agent communications:

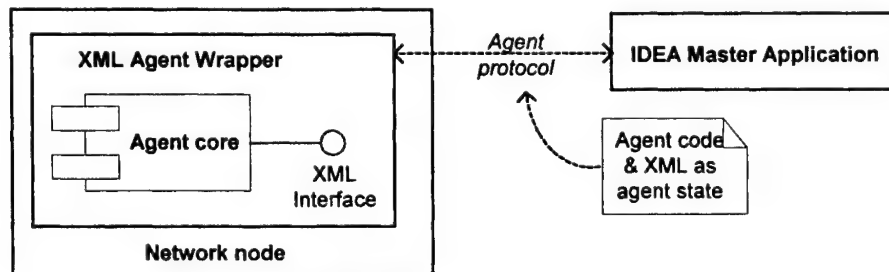


Figure 8.2-5 – XML Agent

8.2.3.2 XML from master application to mobile agent

8.2.4 Interface Clashes

As we saw before, if a client and component expected different interfaces, the only real solution was a specific adapter. But there is a deeper problem with fixed (or “frozen”) interfaces. Specifically, a component interface, once published, is not supposed to change¹. This means the number and types of the parameters to each operation are “hard-coded”. This is often held up as one of the best things about components, as stable interfaces make the architecture as a whole much more stable. However, change is inevitable, and interfaces are no exception, so component providers will occasionally introduce a newer version. When this happens they may or may not continue to support the old one (COM literature insists that they do, but there is nothing forcing them to). Ultimately, frozen interfaces cut in both directions: because of the work and coordination required, interfaces are changed less often than they perhaps should (putting more burden on the clients and

¹ Incidentally, both COM and CORBA provide mechanisms for dynamic component interfaces to be discovered at run time and interpreted. However, these are more for the convenience of non-typed clients (scripting languages) than for the ability to change the interfaces at run time. Although it can be done, it is far from trivial.

components to deal with change), and when they are changed, surprise – there is lots of work and coordination for the clients and components.

However, the XML paradigm makes interface clashes much less of a problem. The primary reason is that XML takes the detail of the interface out of the interface protocol. The protocol is merely XML, which is very stable, so there is no real burden with freezing the interface. For an example, look at a typical operation on an interface (in a simplified IDL syntax):

```
interface IEmployeeServer {  
    bool AddEmployee(  
        int      empID,  
        string firstName,  
        string middleName,  
        string lastName,  
        string ssn );  
    ...  
}
```

Figure 8.2-6 – Example of IDL

Both clients and the components that realize this interface are locked into the exact number and type of parameters. If the component developer wanted to add a street address to the employee, he or she would have three choices:

- Coordinate with all current clients and other components that realize the interface, and add the address parameters to the original operation. All clients and components would need updating.
- Add a new operation like "AddEmployeeWithAddress" – this would not break clients, but they would all have to be recompiled. Components, however, would need to support the new operation. This technique is essentially a hack – it would quickly make the interface complex and hard to maintain.
- Add a new interface, for example "IEmployeeServer2", containing everything that is in IEmployeeServer, with the one changed operation. This causes the least pain initially in that neither existing clients nor existing components would need to do anything – only clients and components that wanted to use the new interface would change. However, they would probably need to write code to deal with both interfaces. The problems with this solution are more long-term: obviously it can't be done very often (when Microsoft supercedes any of its published interfaces, if at all, they only do it once), and it creates a maintenance problem that continues indefinitely.

By contrast, here is an XML interface:

```
interface IXMLControlInterface {  
    /* Returns result as XML doc */  
}
```

```
string AcceptCommand(
    string xmlDoc );
...
```

Figure 8.2-7 – Example XML Interface

and the equivalent command, sent as an XML document:

```
<?xml version="1.0" ?>
<!DOCTYPE EMPLOYEE_SERVER SYSTEM "Employee_Server.dtd">
<COMMAND_ADD_EMPLOYEE>
    <EMPLOYEE ID="12345">
        <NAME>
            <FIRST>Jane</FIRST>
            <MIDDLE>Q.</ MIDDLE>
            <LAST>Doe</LAST>
        </NAME>
        <SSN>123-45-6789</SSN>
    </EMPLOYEE>
</COMMAND_ADD_EMPLOYEE>
```

Figure 8.2-8 – Example XML Document

In this example, the street address could be added to the DTD (as a non-mandatory field), and each client and component could upgrade at any time, or not at all. Here are the effects:

- Non-upgraded clients would not need to do anything. They could continue to send the old command format.
- Upgraded clients would simply send the new command format. They would not need to know if the component they are talking to has been upgraded or not.
- Non-upgraded components would not need to do anything. If they are sent a command with an address, typical XML practice is to disregard any unexpected data.
- Upgraded components could look for an address, and if they find none, they could just enter a blank one. Alternatively, an adapter could be placed between clients and components as discussed earlier. This adapter would add street address fields to any commands that needed them, removing the burden from the upgraded components.

This flexibility of vocabulary is inherent in XML and is probably the primary reason for its popularity.

In summary, the simplicity, flexibility, and wide support of XML make it a natural choice for gluing together complex systems out of heterogeneous components. It is a powerful yet easy to use tool for creating robust software architectures.

9 Appendix C– Prototype Instructions

9.1 Using DTS to Import Scanner Test Data from an Access Database Into an MSDE Database

This section describes how to import STAT® Scanner Test Data from the scanner's Access database into IDEA's MSDE database.

1. Start the DTS import wizard.
2. Set the Access database as the source database.
3. Enter the path to the Access database and click "Next".

The screenshot shows the 'DTS Wizard' window with the title bar 'DTS Wizard' and a close button. The main heading is 'Choose a Data Source'. Below it, a text box says: 'Where would you like to copy data from? You can copy data from any of the sources listed below. Choose one of the following sources.'

Under the heading 'Source:', there is a dropdown menu showing 'Microsoft Access' with a magnifying glass icon. To the left of this is a small icon of a floppy disk.

Below the dropdown, a text box contains the following instructions: 'In order to connect to "Microsoft Access", you must first choose a Database. You may need to provide a valid user name and password. The database password can be specified in the Advanced options.'

Below the instructions are three input fields: 'File name:' with the value 'J:\SDF\Option 2\Test Data\db1.mdb' and a browse button (...); 'Username:' with an empty field; and 'Password:' with an empty field.

At the bottom right of the input fields is a button labeled 'Advanced...'. At the bottom of the dialog are three buttons: '< Back', 'Next >', and 'Cancel'.

4. Set the Destination database as Microsoft OLE DB Provider for SQL Server.

5. Create a new database with the name "testdata" and click "Next".

The screenshot shows the 'DTS Wizard' window at the 'Choose a Destination' step. The title bar reads 'DTS Wizard'. Below the title bar, the text says 'Choose a Destination' and 'Where would you like to copy data to? You can copy data to any of the destinations listed below. Choose one of the following destinations.' A dropdown menu labeled 'Destination:' shows 'Microsoft OLE DB Provider for SQL Server'. Below this, a text box says 'In order to connect to Microsoft SQL Server, you must specify the server, user name and password.' There are three input fields: 'Server:' with '(local)' in the dropdown, 'Username:' with 'sa' in the text box, and 'Password:' which is empty. There are also 'Database:' dropdown with 'testdata' and 'Refresh' and 'Advanced..' buttons. At the bottom are '< Back', 'Next >', and 'Cancel' buttons.

6. Choose "Copy table(s) from source database" and click "Next".

The screenshot shows the 'DTS Wizard' window at the 'Specify Table Copy or Query' step. The title bar reads 'DTS Wizard'. Below the title bar, the text says 'Specify Table Copy or Query' and 'Specify whether to copy one or more tables or the results of a query from the data source.' There are two icons: a floppy disk icon labeled 'Microsoft Access' and a cylinder icon labeled 'Microsoft SQL Server'. Below these, there are three radio button options: 'Copy table(s) from the source database' (which is selected), 'Use a query to specify the data to transfer', and 'Transfer objects and data between Microsoft SQL Server 7.0 databases'. At the bottom are '< Back', 'Next >', and 'Cancel' buttons.

7. Check all three tables and click "Next".

Source Table	Destination Table	Transform
<input checked="" type="checkbox"/> STAT Scanner-produced Vulnerabilities	[testdata].[dbo].[STAT ...]	...
<input checked="" type="checkbox"/> Taxonomy to IAT	[testdata].[dbo].[Taxon...]	...
<input checked="" type="checkbox"/> Tool to Taxonomy	[testdata].[dbo].[Tool to...]	...

< Back Next > Cancel

8. Make sure "Run Immediately" is checked and click "Next".

When

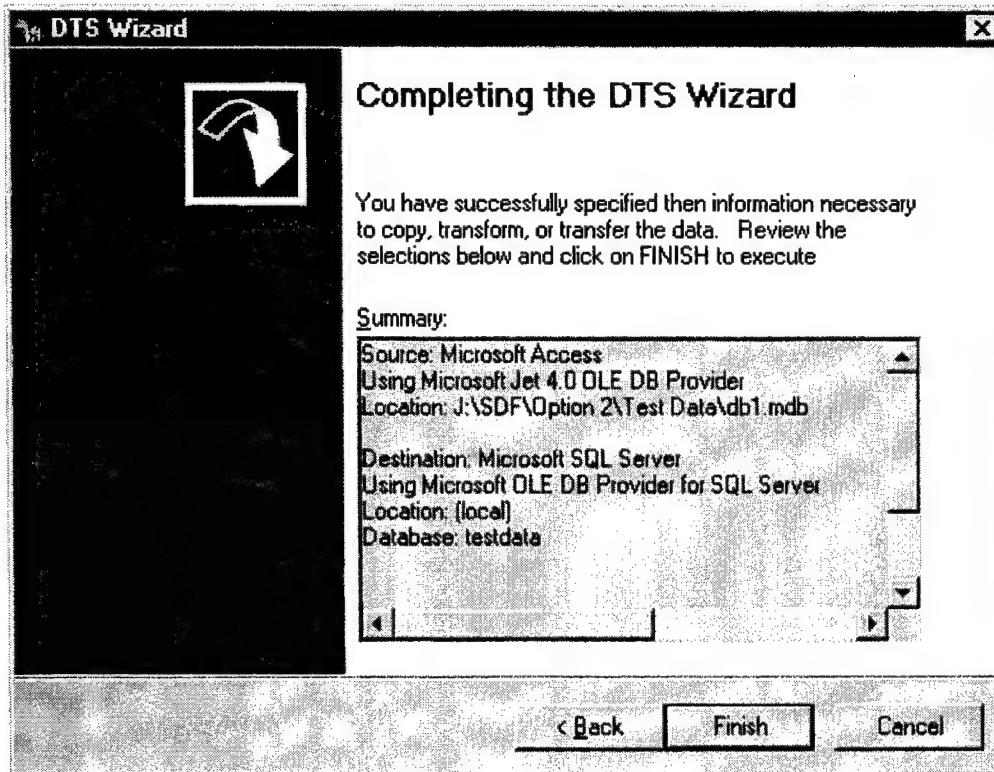
☒ Run immediately ☐ Create DTS package for replication
☐ Schedule DTS package for later execution

Save

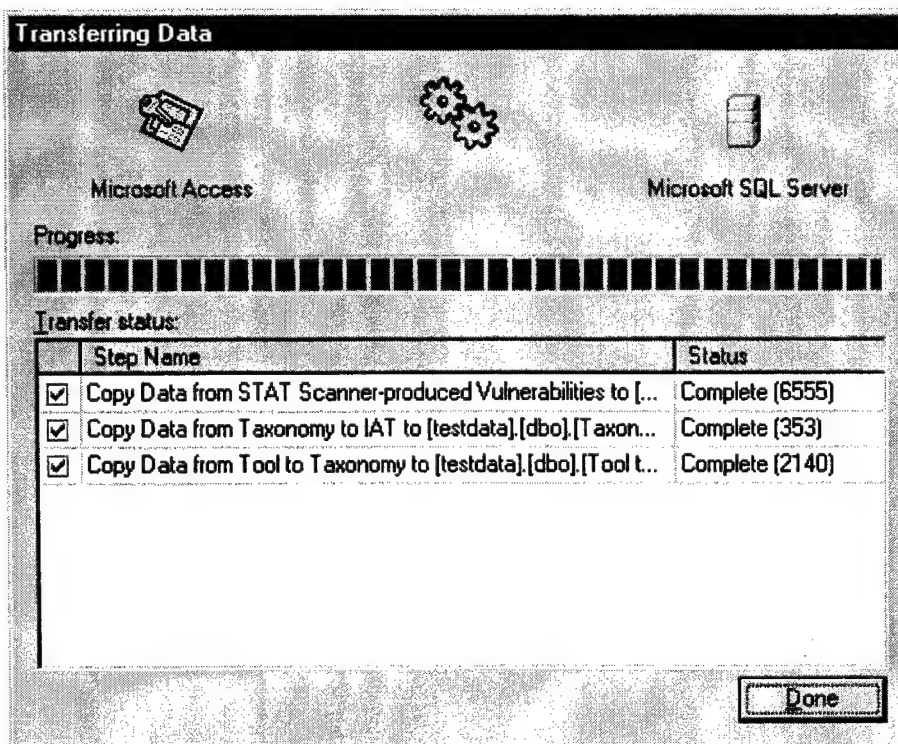
☐ Save DTS Package ☒ SQL Server
☐ Repository
☐ File

< Back Next > Cancel

9. Click "Finish".



10. If all operations have a green check next to them, as follows, then the import was successful.

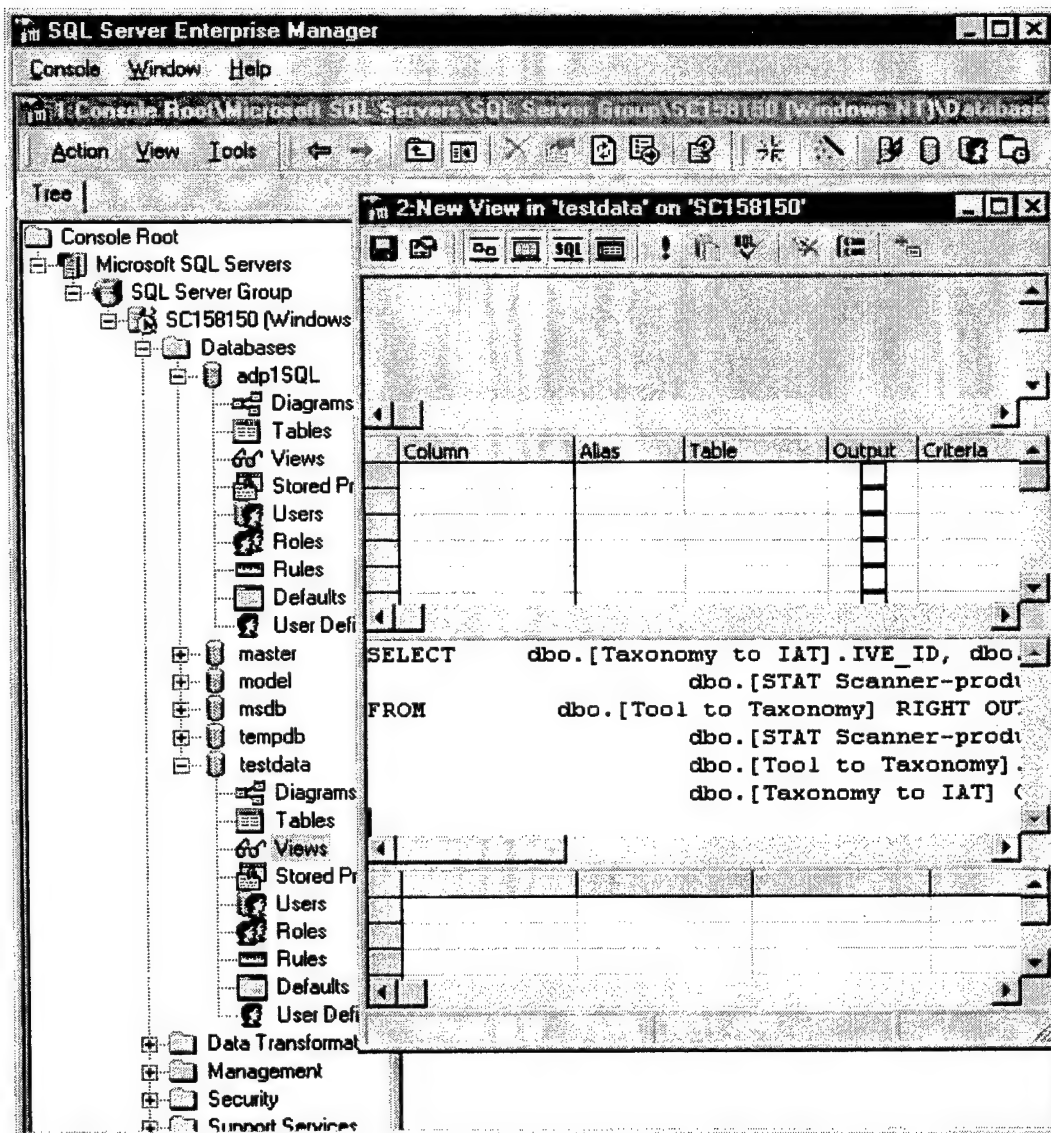


11. Click "Done" to exit.

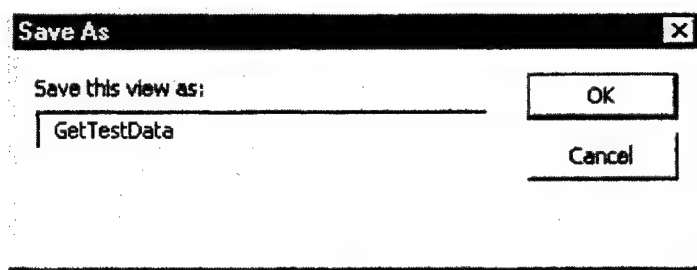
12. Open SQL Enterprise Manager.
13. Register the SQL Server if it has not yet been registered.
14. Expand the tree until you find the database named "testdata".
15. Expand the tree again and find "Views".
16. Right click on "Views" and select "New View".

17. Cut and paste the following SQL statement into the SQL window.

```
SELECT dbo.[Taxonomy to IAT].IVE_ID, dbo.[STAT Scanner-produced
Vulnerabilities].[CVE ID], dbo.[STAT Scanner-produced
Vulnerabilities].[Machine Name], dbo.[STAT Scanner-produced
Vulnerabilities].[Scan Date]
FROM dbo.[Tool to Taxonomy] RIGHT OUTER JOIN
    dbo.[STAT Scanner-produced Vulnerabilities] ON
    dbo.[Tool to Taxonomy].[Tool Vulnerability] = dbo.[STAT Scanner-
produced Vulnerabilities].[Tool Vulnerability] RIGHT OUTER JOIN
    dbo.[Taxonomy to IAT] ON dbo.[Tool to Taxonomy].Taxonomy =
    dbo.[Taxonomy to IAT].Taxonomy
```



18. Save the new view as "GetTestData".



9.2 IDEA XML Generator

This section describes using the IDEA XML Generator to generate test XML for the FuzzyFusion™ component.

Step 1.

Enter the correct connection settings and click the "Connect" button.

The screenshot shows the 'IDEA XML Generator' dialog box. It is divided into several sections:

- Database Connection Settings:** This section is circled in the image. It contains four text input fields: 'Server Name' (containing '(Local)'), 'Database' (containing 'testdata'), 'User Name' (containing 'sa'), and 'Password' (empty). Below these fields are two buttons: 'Connect' and 'Disconnect'.
- Selection Criteria:** This section contains two dropdown menus: 'Node Name' and 'Scan Date'.
- Output Criteria:** This section contains an 'Output File' text input field (containing 'C:\temp\out1.xml'), a checkbox for 'Random Data' (which is unchecked), and two text input fields for 'POE' and 'Trust'.
- Status:** A large text area at the bottom left for displaying status messages.
- Buttons:** 'OK' and 'Exit' buttons are located at the bottom right.

Step 2.

Select the "Node Name" and
"Scan Date".

IDEA XML Generator

Database Connection Settings

Server Name
(Local)

Database
testdata

User Name
sa

Password

Selection Criteria

Node Name
NT43C1

Scan Date
6/3/01 3:09:00 PM

Output Criteria

Output File
C:\temp\out1.xml

☐ Random Data

☐ POE ☐ Trust

Status
Database connection established

Step 3.

Enter the path for the output file and either enter "POE" and "Trust" values or check the "Random" box.

IDEA XML Generator

Database Connection Settings

Server Name
(Local)

Database
testdata

User Name
sa

Password

Connect Disconnect

Selection Criteria

Node Name
NT43C1

Scan Date
6/3/01 3:09:00 PM

Output Criteria

Output File
C:\temp\out1.xml

☒ Random Data

POE Trust

Status

XML written to C:\temp\out1.xml

OK Exit

Step 4. Click "OK" to create XML output.

Note: If you have not yet followed the procedure for importing test data then this program will not run. See the document "Using DTS to Import Scanner Test Data from an Access Database Into an MSDE Database.doc".

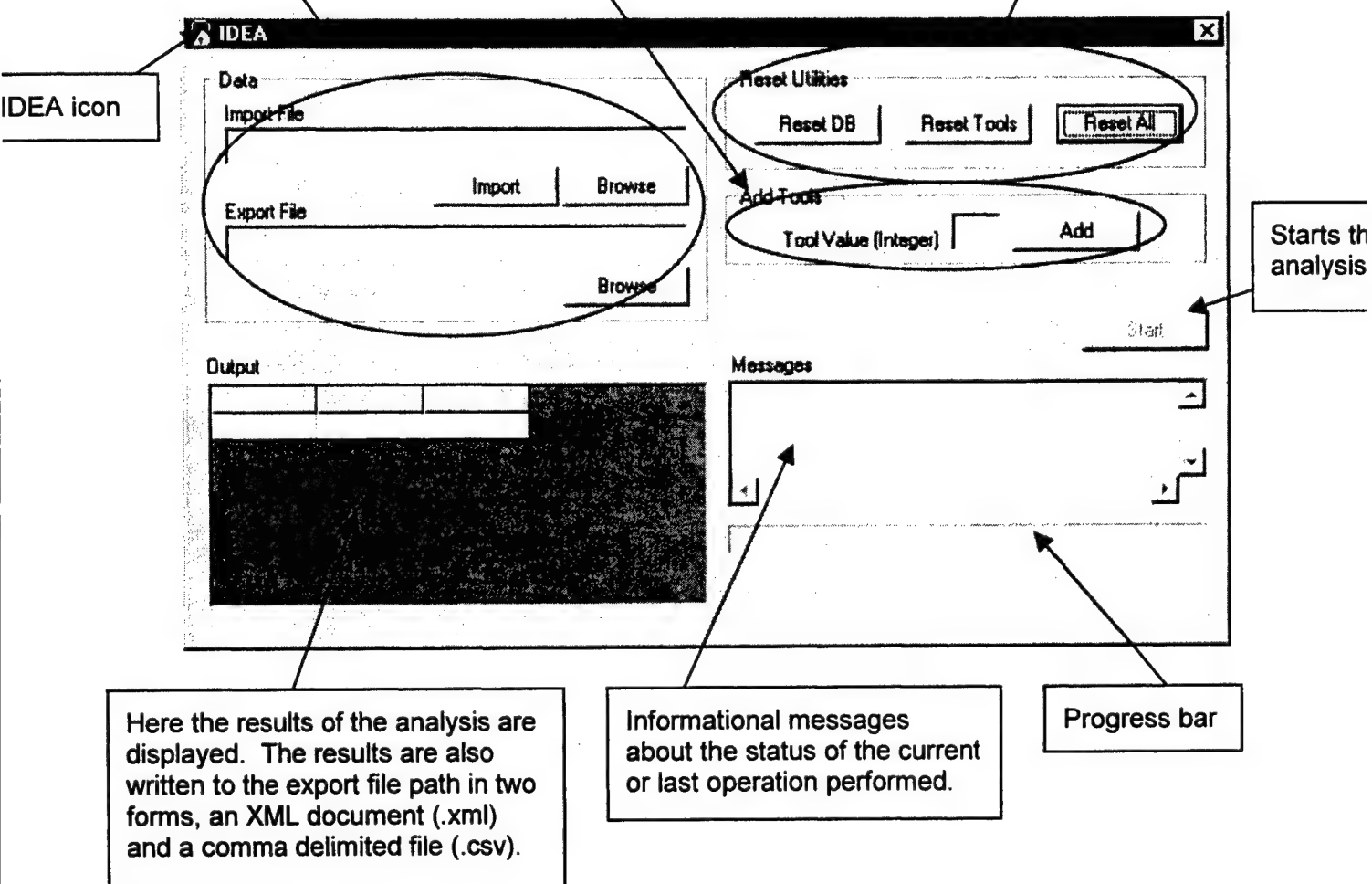
9.3 IDEA Demo Interface

Overview of the IDEA Demo GUI

- **Import File:** Browse for an input data file the choose import to upload data into the IDEA database for analysis.
- **Export File:** Path to place the export file in.

Here additional tools can be added to those included in the import data for analysis to be run on. Up to a three-digit integer value.

- **Reset DB:** Resets the database tables used to store imported data for analysis.
- **Reset Tools:** Resets the list of tool stored in memory, used for analysis
- **Reset All:** Performs all the function of both Reset DB and Reset Tools.



10 Appendix D – IDEA Architecture

The IDEA Architecture document produced during the initial effort is attached.

11 Appendix E – Meta-Language

The Meta-Language document, is attached.

12 Appendix F – IDEA: An Information Superstructure

The Information Superstructure document is attached.

Appendix D

IDEA Software Architecture Document

Table of Contents

1	Scope	1
2	Graphical Conventions Used in this Document.....	2
2.1	Package.....	2
2.2	Rose Model Design Notes	2
2.3	Package Dependency or Instantiation.....	2
2.4	Use Case	3
2.5	Sequence Diagrams	3
2.6	Class Diagrams	3
2.7	Collaboration Diagrams.....	4
2.8	State Diagrams.....	4
2.9	Activity Diagrams.....	5
3	Architecture.....	6
3.1	Software Architecture.....	6
3.2	System Architecture	16
4	System Analysis Use Cases	17
4.1	Use Case Launch Tool.....	18
4.2	Use Case Launch Discover Tool	20
4.3	Use Case Launch Scan Tool.....	27
4.4	Use Case Launch Analysis Tool.....	35
4.5	Use Case Launch Fuzzy Fusion Analysis.....	43
4.6	Use Case View Results	51
5	Repository Model Use Cases.....	57
5.1	Use Case Get Session List	58
5.2	Use Case Open Session	62
5.3	Use Case Close Session	66
5.4	Use Case Manage Session	68
6	Model Preparation Use Cases	73
6.1	Use Case Configure Tool	74
6.2	Use Case Assign Tool.....	76
6.3	Use Case Assign Profile	78
6.4	Use Case Prompt Analyst.....	80
7	Repository Sessions.....	81
7.1	Schemas.....	81
8	Logical View	83
8.1	Threads - Component-Level Use Cases.....	83
8.2	Use Case to Thread Mapping	83
8.3	IDEA API Threads	84
8.4	Repository API Threads	99
8.5	Tool API Threads	113
8.6	Software Design	119
9	Deployment View.....	133
10	Size and Performance	134
11	Security.....	135
11.1	Database Security	135
11.2	Security Mechanisms.....	136
11.3	Authenticating Database Users with Windows NT	139
12	Appendix A	144
12.1	CSM minimum information	144
12.2	Taxonomies	144
13	Appendix B.....	145
13.1	Table of CSM Layers	145
14	Appendix C.....	146
14.1	Table of IA Tool API Capabilities.....	146
15	Glossary.....	147

List of Tables

Table 4-1 - Launch Discover Tool Scenario 1	21	Table 5-6 - Open Session Scenario 3	65
Table 4-2 - Launch Discover Tool Scenario 2	22	Table 5-7 - Open Session Scenario 4	65
Table 4-3 - Launch Discover Tool Scenario 3	23	Table 5-8 - Close Session Scenario 1	67
Table 4-4 - Launch Discover Tool Scenario 4	24	Table 5-9 - Close Session Scenario 2	67
Table 4-5 - Launch Discover Tool Scenario 5	25	Table 5-10 - Manage Session Scenario 1	69
Table 4-6 - Launch Scan Tool Scenario 1	30	Table 5-11 - Manage Session Scenario 2	70
Table 4-7 - Launch Scan Tool Scenario 2	31	Table 5-12 - Manage Session Scenario 3	70
Table 4-8 - Launch Scan Tool Scenario 3	31	Table 5-13 - Manage Session Scenario 4	70
Table 4-9 - Launch Scan Tool Scenario 4	32	Table 5-14 - Manage Session Scenario 5	71
Table 4-10 - Launch Scan Tool Scenario 5	33	Table 5-15 - Manage Session Scenario 6	71
Table 4-11 - Launch Analysis Tool Scenario 1 ..	38	Table 5-16 - Manage Session Scenario 7	72
Table 4-12 - Launch Analysis Tool Scenario 2 ..	39	Table 7-1 - Layer	81
Table 4-13 - Launch Analysis Tool Scenario 3 ..	39	Table 7-2 - Node	82
Table 4-14 - Launch Analysis Tool Scenario 4 ..	40	Table 7-3 - Attributes	82
Table 4-15 - Launch Analysis Tool Scenario 5 ..	41	Table 8-1	83
Table 4-16 - Launch Fuzzy Analysis Tool		Table 8-2	85
Scenario 1	45	Table 8-3	88
Table 4-17 - Launch Fuzzy Analysis Tool		Table 8-4	88
Scenario 2	46	Table 8-5	90
Table 4-18 - Launch Fuzzy Analysis Tool		Table 8-6	92
Scenario 3	47	Table 8-7 Control Parameters	95
Table 4-19 - Launch Fuzzy Analysis Tool		Table 8-8	95
Scenario 4	48	Table 8-9	98
Table 4-20 - Launch Fuzzy Analysis Tool		Table 8-10	98
Scenario 5	49	Table 8-11	100
Table 4-21 - View Results Scenario 1	52	Table 8-12	103
Table 4-22 - View Results Scenario 2	54	Table 8-13	104
Table 4-23 - View Results Scenario 3	55	Table 8-14	105
Table 5-1 - Get Session List Scenario 1	59	Table 8-15	107
Table 5-2 - Get Session List Scenario 2	59	Table 8-16	113
Table 5-3 - Get Session List Scenario 3	60	Table 8-17	114
Table 5-4 - Open Session Scenario 1	63	Table 14-1 - Tool configuration	146
Table 5-5 - Open Session Scenario 2	64		

List of Figures

Figure 2-1 - Package Example.....	2	Figure 4-24 - Execute Fuzzy Fusion Analysis Interaction Diagram	45
Figure 2-2 - Note Example	2	Figure 4-25 - Basic flow	46
Figure 2-3 - Package Dependency or Instantiation.....	2	Figure 4-26 - No fuzzy analysis tool available....	47
Figure 2-4 - Use Case	3	Figure 4-27 - Fuzzy analysis tool is not properly configured	48
Figure 2-5 - Use Case with Interface	3	Figure 4-28 - Unable to update session.....	49
Figure 2-6 - Sequence Diagram	3	Figure 4-29 - Unable to format results	50
Figure 2-7 - Example Class Diagram	4	Figure 4-30 - View Results Use Case	51
Figure 2-8 - Collaboration Diagram	4	Figure 4-31 - View Results Preliminary Interaction Diagram.....	52
Figure 2-9 - State Diagram	4	Figure 4-32 - Basic flow	53
Figure 2-10 - Activity Diagram	5	Figure 4-33 - Unable to print	55
Figure 3-1 - Component Architecture.....	6	Figure 4-34 - Unable to save	56
Figure 3-2 - Architecture Dependencies.....	8	Figure 5-1 - Data Model Diagram.....	57
Figure 3-3 - Architecture Design.....	9	Figure 5-2 Get Session Lists Use Case	58
Figure 3-4 - IDEA Component Architecture	10	Figure 5-3 - Basic Flow	59
Figure 3-5 - Repository Component Architecture	11	Figure 5-4 - No sessions available	60
Figure 3-6 - Tool Component Architecture	12	Figure 5-5 - Unable to find a database	61
Figure 3-7 - Communication Component Architecture	13	Figure 5-6 - Open Session.....	62
Figure 3-8 - Component Detailed View	15	Figure 5-7 - Basic flow	64
Figure 4-1 - Launch Tool Use Case.....	18	Figure 5-8 - Close Session Use Case	66
Figure 4-2	19	Figure 5-9 - Manage Session Use Case.....	68
Figure 4-3 - Launch Discover Tool Use Case	20	Figure 6-1 - Configure Tool Use Case	74
Figure 4-4 - Basic Flow	22	Figure 6-2 - Assign Tool Use Case	76
Figure 4-5 - No discovery tools available.....	23	Figure 6-3 - Assign Profile Use Case.....	78
Figure 4-6 - Discovery tool is not properly configured.....	24	Figure 6-4 - Prompt Analyst Use Case.....	80
Figure 4-7 - Unable to update session	25	Figure 7-1 - Data Model Diagram.....	81
Figure 4-8 - Unable to format results.....	26	Figure 8-1 - Open Investigation	89
Figure 4-9 - Scan System Use Case.....	27	Figure 8-2 - Start Investigation	91
Figure 4-10 - Scan System Analysis Interaction Diagram	29	Figure 8-3 - Start Investigation	93
Figure 4-11 - Basic flow	30	Figure 8-4 - Close Investigation.....	96
Figure 4-12 - No scan tools available	31	Figure 8-5 - Close Session	101
Figure 4-13 - Scan tool is not properly configured	32	Figure 8-6 - Create Session	102
Figure 4-14 - Unable to update session	33	Figure 8-7 - Fetch Session List	103
Figure 4-15 - Unable to format results.....	34	Figure 8-8 - Fetch Tool Information	104
Figure 4-16 - Analyze Model Use Case.....	35	Figure 8-9 - Fetch Tool List.....	106
Figure 4-17 - Analyze Model Analysis Interaction Diagram	37	Figure 8-10 - Open Session.....	107
Figure 4-18 - Basic flow	38	Figure 8-11 - Save Session.....	108
Figure 4-19 - No analysis tool available.....	39	Figure 8-12 - View Session.....	110
Figure 4-20 - Analysis tool is not properly configured.....	40	Figure 8-13 - Launch Tool	116
Figure 4-21 - Unable to update session	41	Figure 8-14 - Component Architecture	119
Figure 4-22 - Unable to format results.....	42	Figure 8-15 - IDEA API Class	120
Figure 4-23 - Execute Fuzzy Fusion Use Case ...	43	Figure 8-16 - IDEA Engine Layer	124
		Figure 8-17 - Repository API.....	126
		Figure 8-18 - Repository Engine Layer.....	129
		Figure 8-19 - Tool API.....	130

1 Scope

This document describes the architecture for the Integrated Design Environment for Assurance (IDEA) Program.

Section 2 briefly describes the Unified Modeling Language (UML) conventions used within the rest of the document.

Section 3 provides an overview of the architecture including the architecture styles used for the components.

Section 4 introduces the use cases defined for analyzing systems for vulnerabilities.

Section 5 introduces the use cases for managing the data model.

Section 6 introduces the use cases for related to tool management.

Section 7 describes the Common System Model (CSM) data structure.

Section 8 presents the logical view of the architecture that details the threads, API's and Class design.

Section 9 presents a view of the deployed system.

Section 10 discusses size and performance issues and resolutions.

Section 11 discusses security factors for the architecture.

Appendix A lists the minimum information for a CSM model for each tool and the CSM taxonomies.

Appendix B enumerates the tables defined for the CSM.

Appendix C presents the application programming interfaces (APIs) for the Commercial Off The Shelf Tools (COTS) used for system vulnerability testing.

Appendix D is a glossary of terms.

2 Graphical Conventions Used in this Document

The software architecture model contains use cases, logical views, static views, class diagrams, and other diagrams, all depicting the software architecture. The following sections explain the UML graphics used by the modeling tool.

2.1 Package

A package is used to group one or more UML items (including nested packages).

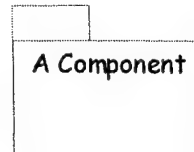


Figure 2-1 - Package Example

2.2 Rose Model Design Notes

Notes look like a page from a note pad and they are used for Displaying text on page. Notes may be attached to virtually any UML graphic being shown and are used to provide additional information or bring attention to the attached item.

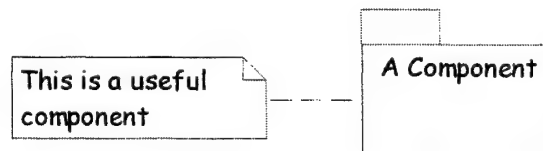


Figure 2-2 - Note Example

2.3 Package Dependency or Instantiation

A dependency or instantiation looks like an arrow with a dashed line; it links one item to another. The following shows a dependency of one package on another package.

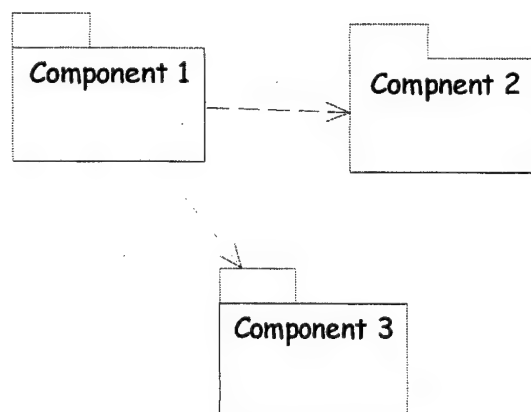


Figure 2-3 - Package Dependency or Instantiation

2.4 Use Case

The actor and use case with associations are shown in a use case graphic as follows:



Figure 2-4 - Use Case

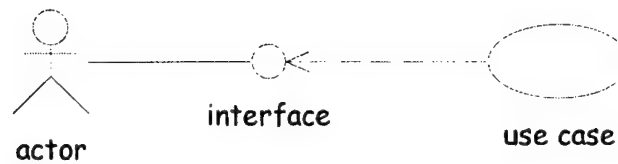


Figure 2-5 - Use Case with Interface

2.5 Sequence Diagrams

Sequence Diagrams show the order of execution between actors and objects or objects and other objects.

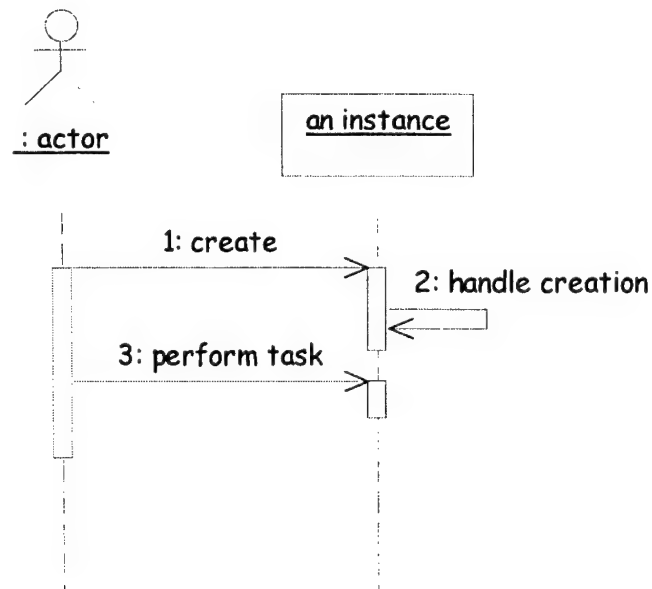


Figure 2-6 - Sequence Diagram

2.6 Class Diagrams

The class diagram shows the static structure of the objects.

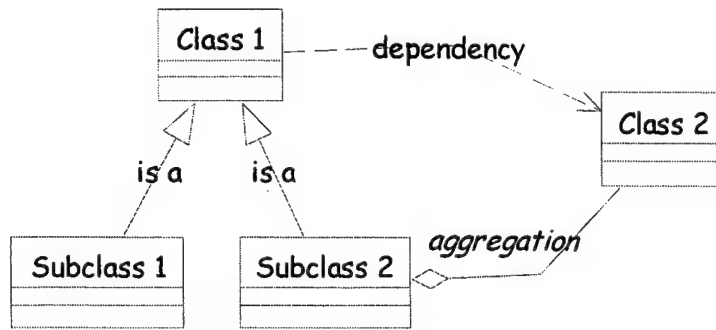


Figure 2-7 - Example Class Diagram

2.7 Collaboration Diagrams

Collaboration diagrams show a network view of interactions (messages sent and data flow) between objects.

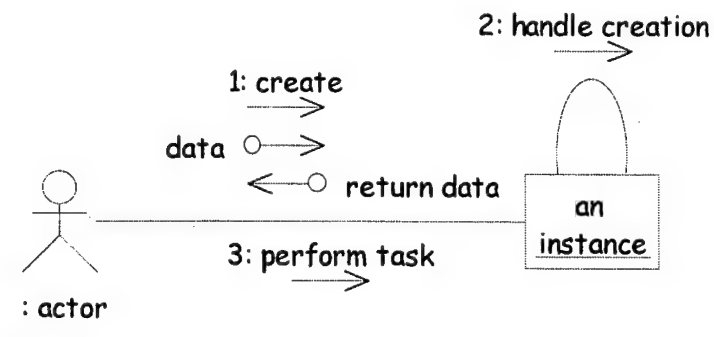


Figure 2-8 - Collaboration Diagram

2.8 State Diagrams

State Diagrams show the states of a given object and the actions that cause transitions between those states.

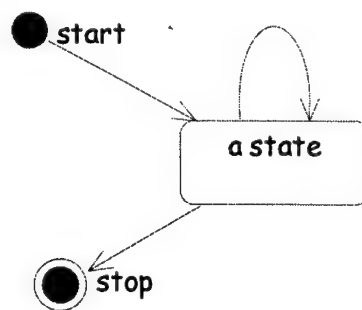


Figure 2-9 - State Diagram

There are documentation attributes for each state and event.

2.9 Activity Diagrams

Activity diagrams show procedural flow and behavior with control structure.

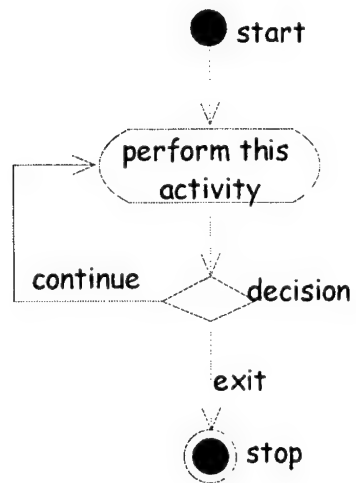


Figure 2-10 - Activity Diagram

3 Architecture

This section provides an overview of the IDEA architecture.

3.1 Software Architecture

This section discusses the architecture style, architecture constraints, and goals of IDEA.

3.1.1 Introduction

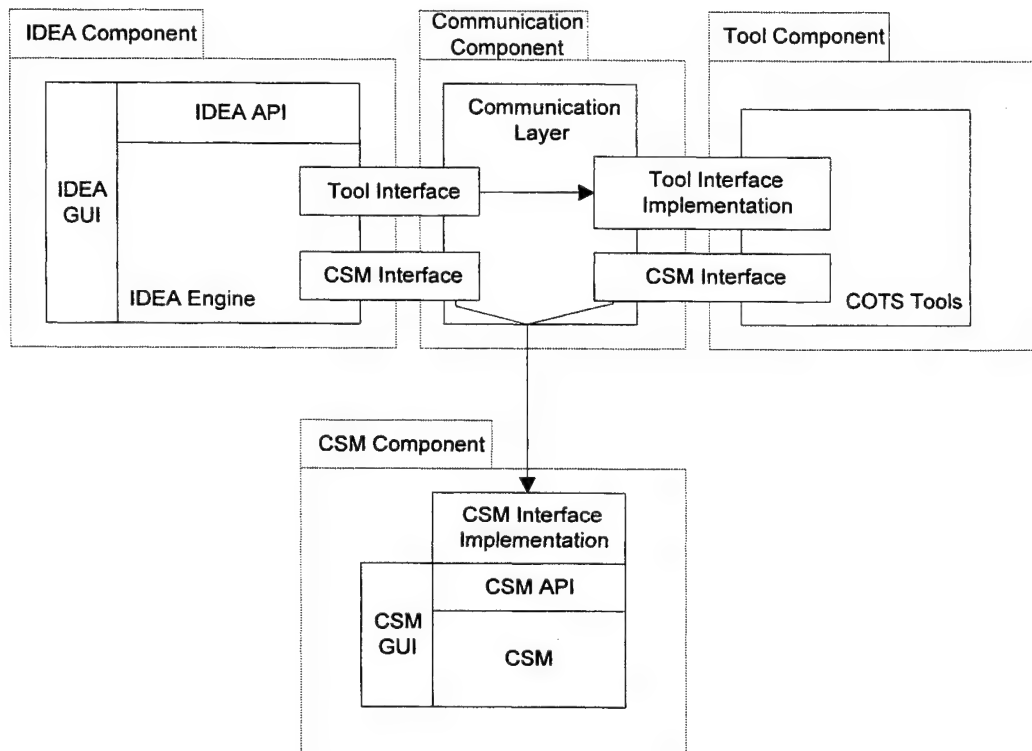


Figure 3-1 - Component Architecture

The IDEA software architecture is a heterogeneous architecture. At the level shown in Figure above, the style is a distributed architecture with the Communication component acting as a Broker between the other three components.

Components separate the individual functionality of each of IDEA's major behaviors.

Each component is designed as a cohesive entity with a well developed scope. The IDEA component allows users to open models and run IA tools against them. The Repository component allows users to develop and manage model templates. The Tool component supports IA COTS tool interaction with the IDEA and Repository component.

The components are loosely-coupled with respect to each other and each has an independent architecture. At the component level (for IDEA, Repository, and Tool), the architecture style exploited is a 3-tier layered architecture.

The design goal behind the components is to have the implementation of their behavior decoupled from the standardized operations available in their interfaces. With the exception of the Communication component,

each contains an interface, engine, and repository layers. These layers directly correspond to the Interface, Business, and Data Storage layers of standard 3-tier architectures.

Another style used in the architecture is a Repository style. This style is a database-centric architecture where the repository is used to maintain states of the components. The flavor of Repository style used is that the IDEA component maintains the triggers and controls for the repository. With some database implementations it is possible to imbed triggers and other control logic within the repository. The database imbedded approach was not chosen since it may limit the architecture to particular repository implementations.

Here are some rules governing the design of the system:

- Communication between each component is managed by the Communication component. The Communication component will act as a *broker* between the other components.
- Each component has a well-defined interface that supports the operations necessary for the other components. There is some risk here associated with the completeness of the interfaces.

To mitigate the impact of the complexities of component interface layers, the interfaces are split into usage categories. For example, the Repository interface is separated into an interface to handle IDEA component requests, an interface to handle Tool component requests, and an interface to handle Repository GUI requests. The other component interfaces are similarly categorized.

IDEA's interface has several scopes. It is intended that all GUI calls go through an IDEA API. The second interface scope is for other external components. Primarily, and probably solely, other external components will be limited to the Tool component.

The Repository interface supports interaction between the database and the IDEA and Tool components as well as its own Repository GUI. IDEA uses the Repository interface to get lists of sessions within the database. The Tool component uses the Repository interface to get model information for the tool's use and to store tool results.

- All component interactions flow through the Business and Repository layers to the Interface layers of other components. With the exception of the Communication component, each component implements this 3-tiered architecture.

The Communication component is a two-tier layer component with no data service layer. The communication interface layer simply brokers the communication requests. All communication configuration information is stored in the Communication component's Logic layer.

- Each component's repository layer is nothing more than a proxy to the database. One implementation strategy for these layers is that they are nothing more than calls to the Repository component's interface. It may also be the case that the implementation of this layer is a real repository managing the data necessary for the individual components. Certainly component configuration information would be stored and maintained within the repository layer.

Each component's Repository layer stores data for the component. The Repository layer for the IDEA and Tool components is implemented as a proxy pattern to the component's data within the database. This design implies that the Repository component services all of the persistent data requirements for the IDEA and Tool components.

The Tool interface supports launching discovery, scanner, analysis, and Fuzzy Fusion¹ tools.

Overall, the design is a database-centric, distributed-component architecture with the major components interacting through a message broker. With the exception of the Communication component, each component

¹ Fuzzy Fusion is a Trademark of Harris Corporation.

is designed as a three-tier architecture. The four components, IDEA, Repository, Tool, and Communication require well-defined interfaces for this goal. All interactions with a component are handled by the component's API. Business logic for each component is stored within the component's Engine layer.

Figure 3-2 shows the dependencies between the component's layers. Within the components, the interface and business layers interact. The business layers use the internal data layer for data storage and retrieval. Inter-component communication is limited to business-to-interface communication or, in the case of the data layer, data-to-interface communication. This standard is followed throughout this document.

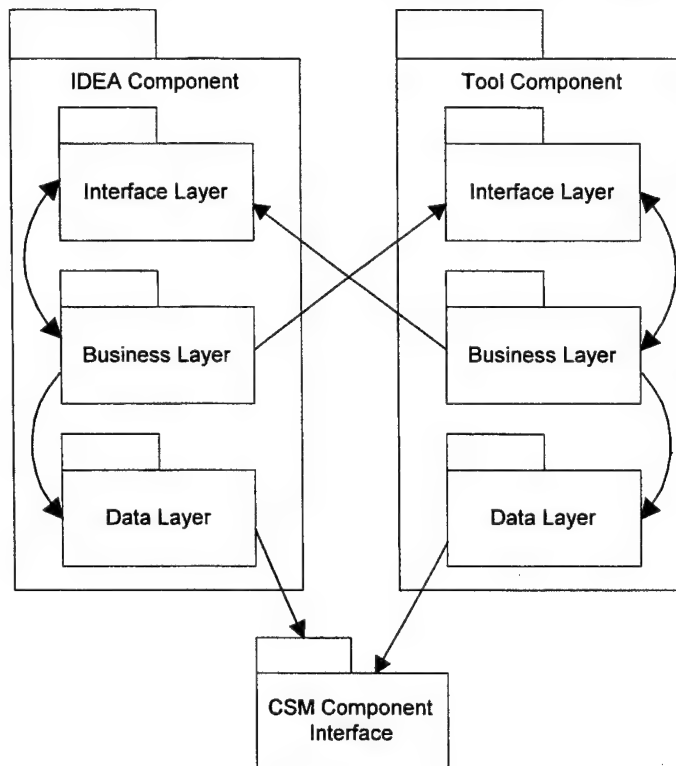


Figure 3-2 - Architecture Dependencies

The rest of this section discusses the scope of each of the component layers.

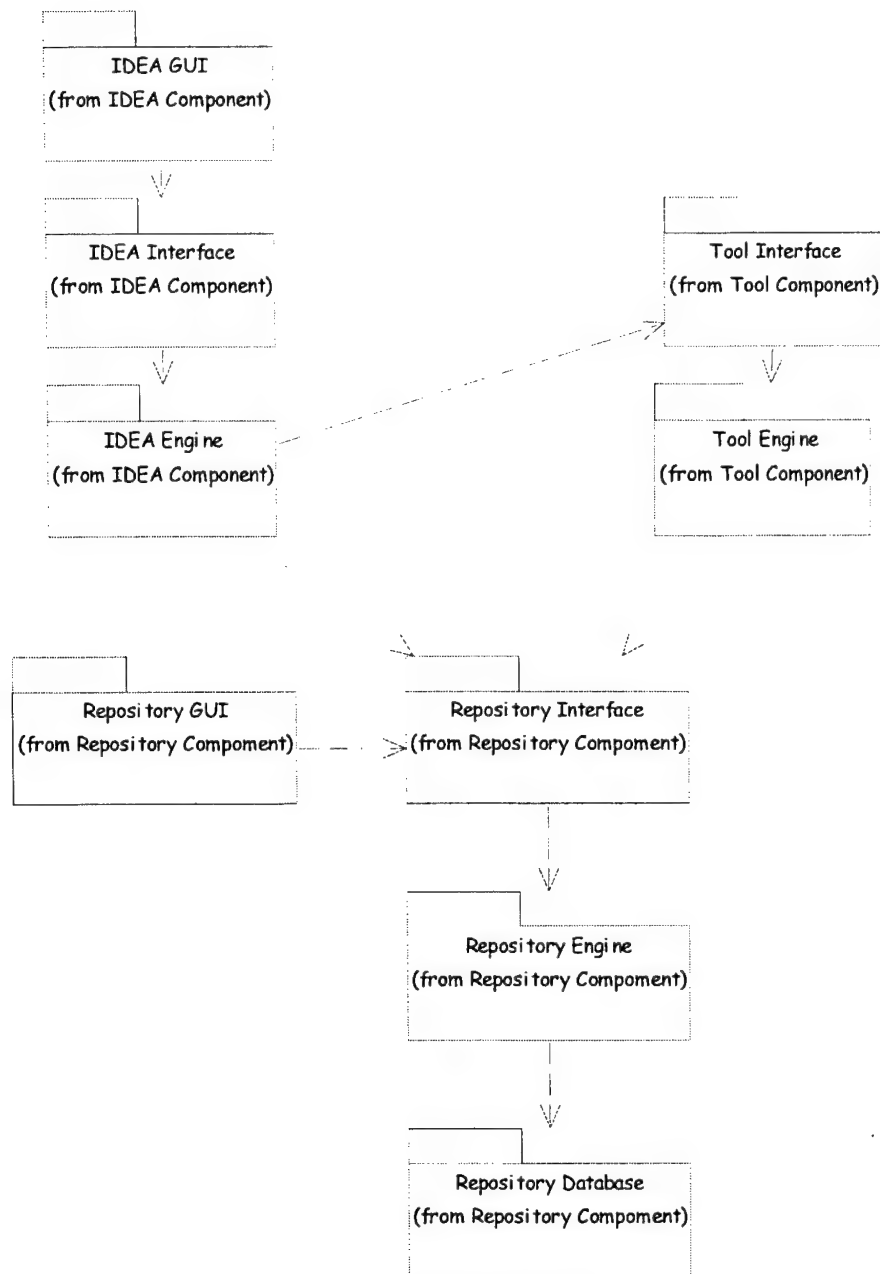


Figure 3-3 - Architecture Design

3.1.2 IDEA Component

The IDEA component supports behaviors related to opening sessions, launching tools against a model, and viewing tool results. The intent of this design is to isolate management logic for session control within the IDEA Engine layer. The IDEA Interface layer is split into a GUI and API section. It is intended that all interaction with the IDEA Engine layer occur through calls to the API. However, for the GUI, this may not be efficient. For external components it is required that they only communicate to the IDEA Engine through the IDEA API.

The IDEA component allows users to open Repository sessions, launch tools against the sessions, view, and print tool results.

Through a user's request, the IDEA component gets a list of tools to launch against a model and then launches them. This logic is within this component to allow the flexibility to launch tools independently of one another without having to define a tool manager within the Tool component. If tools are independent of one another, in terms of model information and tool result information, it may be possible to configure IDEA to launch multiple tools or tool sets at once. The rationale for having IDEA launch tools is to separate the tool control logic with the tool execution logic. IDEA controls the tool launching. Each tool defines the meaning behind 'tool launching.' See Tool Component Design for elaboration on the Tool component's scope.

The following use cases drive the protocol for the IDEA component interface.

3.1.2.1 Design

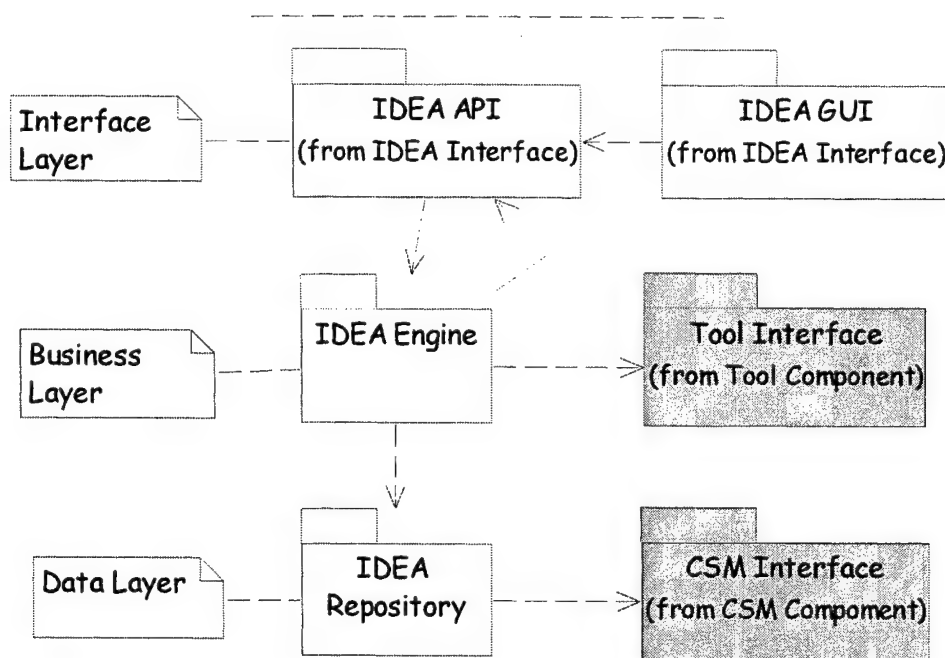


Figure 3-4 - IDEA Component Architecture

The IDEA Component is composed of three cohesive layers: an interface layer, a business unit layer, and a repository (or data) layer.

The design of this component separates user's (GUI users and API users) requests from the commands and logic necessary to implement the user's requests.

The interface layer of the IDEA Component is composed of a GUI and an Application Programming Interface (API). The IDEA Engine implements the business layer and IDEA Storage implements the repository layer.

The interface layer supports all IDEA Component interactions. The interface layer provides options for interacting with users as well as other software. Users of the IDEA interact with the GUI to define and analyze their systems. The IDEA API supports all GUI commands as well as a set of commands available to external IA tools.

The business layer supports session management, tool configuration, and tool execution. IDEA Engine processes requests from the API through internal logic. Its data needs are supported by IDEA Storage.

Information for launching tools as well as general session information is acquired by querying IDEA Storage. IDEA Engine communicates externally to the Tool Component for launching tools.

IDEA Storage may be implemented as a set of *proxy* objects to the actual data.

3.1.3 Repository Component

The Repository Component is the repository for the IDEA and Tool components. The intent of the Repository component design is to have an interface to the database and a layer to implement logic that may be required to convert interface requests to repository logic.

When the Repository API gets a request for the model information needed by a tool, it will format the information per the tools needs. The tool will receive a custom-formatted version of the model. Formatting the model implies that each tool has some configuration or data mapping information stored within the database. While the execution of the tool may be limited to the Tool component, the converting of CSM model format to Tool format is done within the Repository component.

3.1.3.1 Design

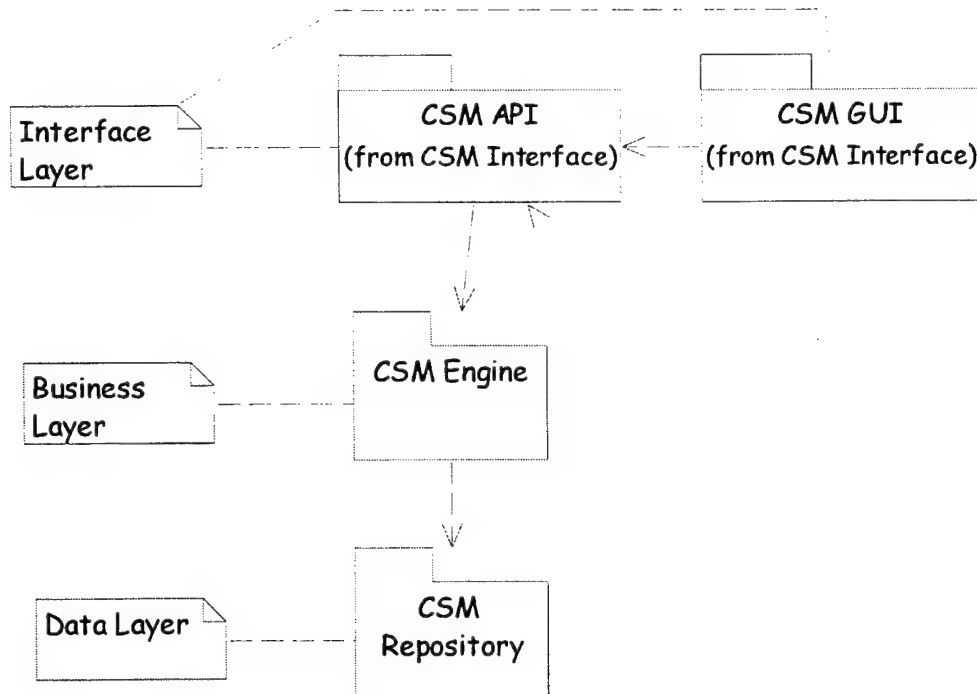


Figure 3-5 - Repository Component Architecture

The Repository Component is separated into three layers. Although this separation may not be carried over into implementation, its design guides the separation of concerns between the parts of the Repository Component.

The design of this component decouples calls to the repository from the actual repository implementation and design. Calls to the repository are mapped to the repository implementation within the Repository Engine.

As with the IDEA component, there are two sections in the Repository component's interface layer. One section is the Repository GUI allowing users to manage the database. The other section is the Repository API

which all external components will use to request or store model information. The interface layer of the Repository enforces a standard protocol for accessing the database by external components.

The business layer of the Repository, Repository Engine, is responsible for handling the possible interpretation of the interface protocol to the repository protocol. Also, the Repository Engine is responsible for converting the database information to the formats understood by the tools.

Whether or not this middle layer is imbedded in repository logic (or in separate code specifically implemented for the job) is unimportant to understanding the design at this point. Suffice it to say that the Repository component performs two functions for the external components. It accepts model queries from the IDEA component and the Tool component. For the IDEA component, it returns information about a Repository session. For the Tool component, it returns a model tailored to the tool's needs.

If the repository implementation supports imbedded procedural logic, the implementation may be a thin interface client and thick repository server. However, this implementation would possibly, as previously stated, limit the COTS tools used to implement the repository.

3.1.4 Tool Component

The Tool component is responsible for launching COTS IA tools and storing of their results. The goal of the Tool component design is to allow tools to be easily plugged in to the component. Through the tool interface layer, the IDEA component launches tools contained on the session's tool list. Each tool is responsible for its own repository data. Each tool asks the Repository Component to give it the model information it requires. Each tool returns its results to the repository. There will be a filter placed between the Tool and database.

3.1.4.1 Design

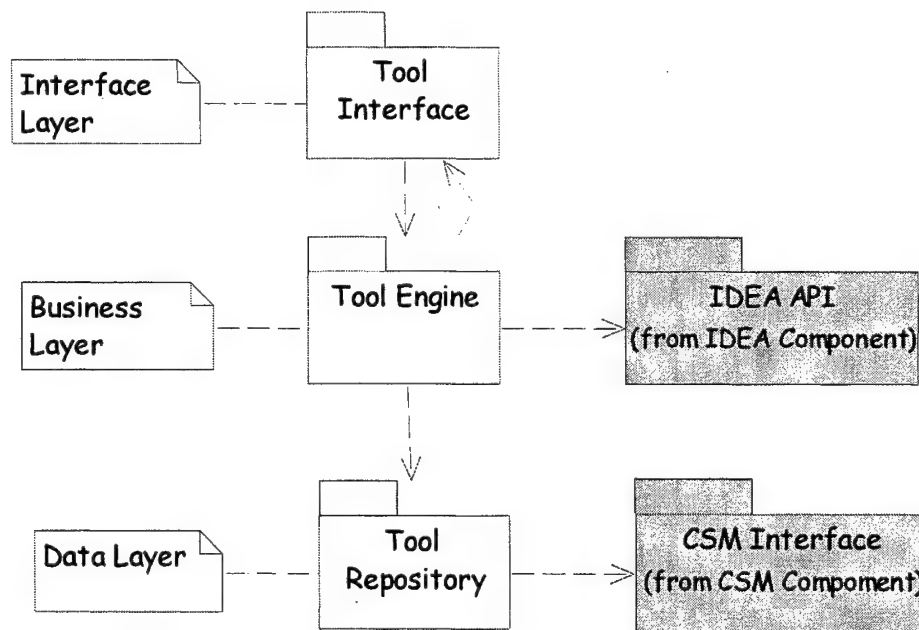


Figure 3-6 - Tool Component Architecture

The Tool Component is separated into three layers.

The Tool Interface provides the necessary control mechanisms for launching tools.

The design of this component provides a separation of tool commands from the actual tools. Tool commands are abstracted to the needs of the users. The actual tool interface, the code that calls the COTS tool is encapsulated within the Tool Engine.

The Tool Engine accepts tool commands from the interface and reacts accordingly. It interfaces with the Repository Component (via Tool Storage) to get information required by a tool as well as storing information generated by tools.

The Tool Storage is a proxy to the Repository Component.

3.1.5 Communication Component

The Communication component is included separately within this architecture as a means of isolating the inter-component communication from the communication implementation. The IDEA, Tool, and Repository components interact with each other via messages. The definition for the message's transport mechanism contained within the Communication component.

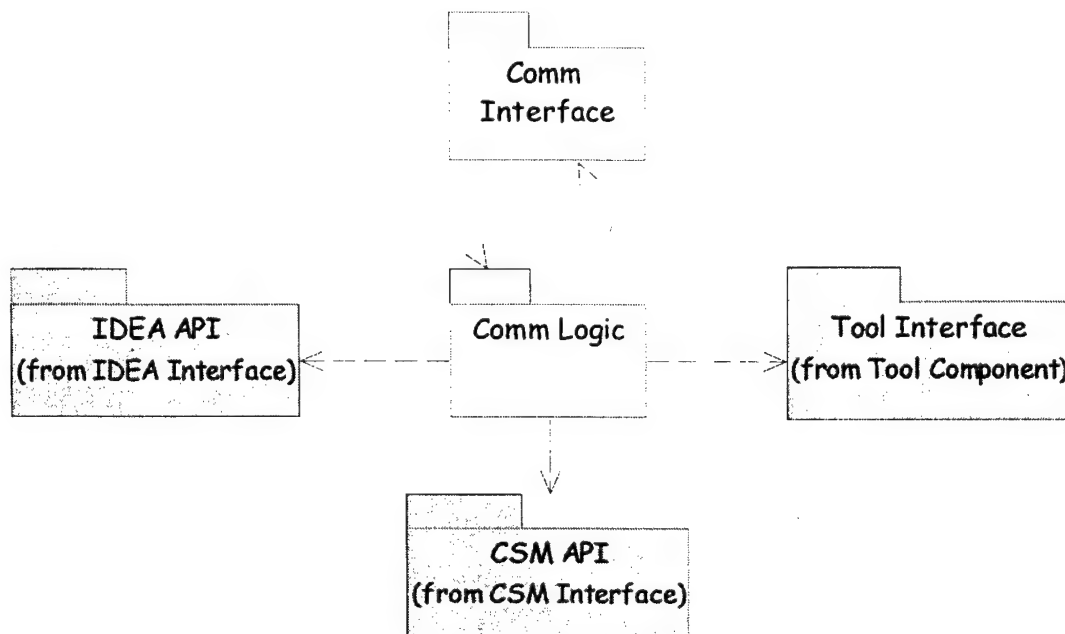


Figure 3-7 - Communication Component Architecture

The design of this component provides a communication abstraction for inter-component messages. Components will interact with one another using the messages provided within the Communication component. The communication component supports the following communication routes between components:

- IDEA to Repository
- IDEA to Tool
- Tool to Repository
- Tool to IDEA

The Repository component does not initiate calls to the IDEA or Tool components.

The Communication component is designed with two layers. The interface layer, named Comm Interface, provides services for message clients. The business layer, named Comm Logic, provides the message service at a server level.

The Comm Interface provides an interface between the caller and the recipient.

The Comm Logic delivers the message to the recipient's interface.

3.1.6 Detailed View

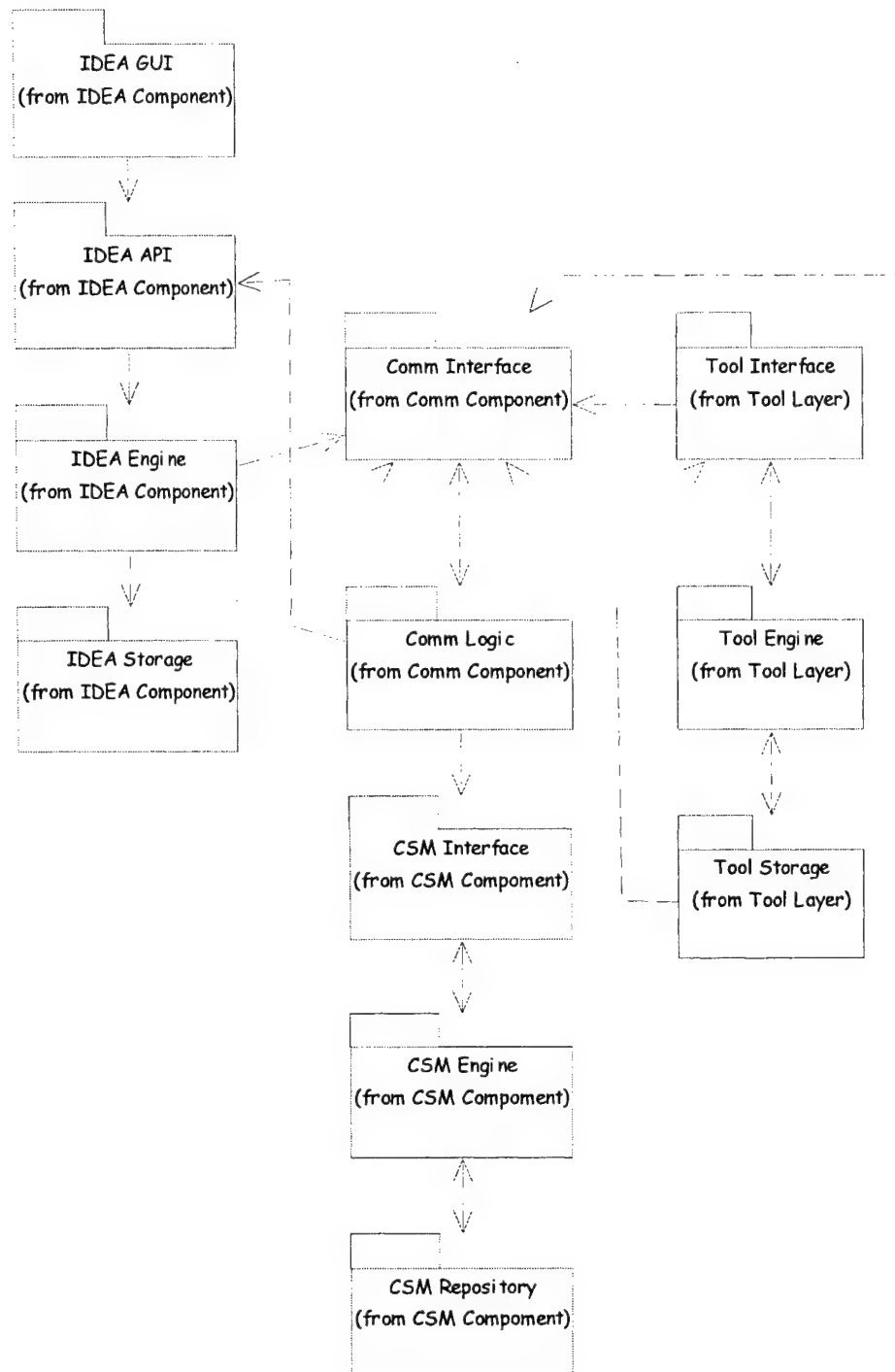


Figure 3-8 - Component Detailed View

3.2 System Architecture

4 System Analysis Use Cases

This section discusses the use cases that support the function of analyzing a system model. The system model is filled with data by discovery tools. The use cases Build System and Discover System are responsible for filling out analysis information for a system model.

The use case Analyze System is the heart of this section. It includes three other use cases: Open Analysis Session, Select Analysis for Launch, and Launch Analysis. From this use case the major functionality of IDEA is defined.

A *session* represents a system model and is stored within the repository. The use case Open Analysis Session supports user interaction and selection of a session to open. After this use case is complete, the Select Analysis for Launch is invoked.

The Select Analysis for Launch use case includes the use case for selecting sessions; the Select Analysis Session use case. The Select Analysis Session use case is presented separately with the intent that it may support several use cases. Once a session is selected, it may be launched, opened for reading, opened for modification, etc. Select Analysis for Launch allows the user to select a session and launch it. The term *launch* means that the session will be checked for vulnerabilities by executing tools to analyze the session information.

The Launch Analysis use case is invoked in response to a user opting to launch an analysis of a session. Its scope is entered immediately after the Select Analysis for Launch use case completes. The Launch Analysis use case is extended by Scan System , Analyze Model, and Execute Fuzzy Fusion.

Scan System is responsible for launching scanner tools against the hardware system and, if needed against the session information. Launching analysis tools against the session information is defined by the Analyze Model use case. The Execute Fuzzy Fusion use case defines how Fuzzy Fusion is used. Each of these three use cases is responsible for launching the COTS tools. The launching of the tools is split into three separate use cases only because there may be significantly different behavior and data between the three tool types.

The final use case of this section is View Results. It defines how the results of the Analyze System use case are made available to the user.

The following sections describe these use cases in detail.

4.1 Use Case Launch Tool

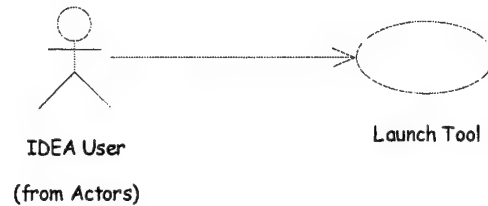


Figure 4-1 - Launch Tool Use Case

4.1.1 Brief Description

In this use case, the IDEA system allows users to run tools against the session information.

4.1.2 Pre-Conditions

Before this use case starts, the session must have been created in the database.

4.1.3 Flow of Events

4.1.3.1 Start of use case

This use case starts when the session has been opened and locked.

4.1.3.2 Basic flow

1. Launch the tool

After the IDEA User selects a session to launch, the session name and tool name is used to launch a tool. The tool sends a query to the database for tool-specific information.

After the tool execution has completed, it stores its results into the database.

2. Store tool results

The tool sends its updated information to the database along with the session information.

4.1.3.3 End of use case

This use case ends when the tool has stored any changes to the session into the repository.

4.1.3.4 Alternative flows

4.1.4 Post-Conditions

4.1.5 Scenario Diagrams

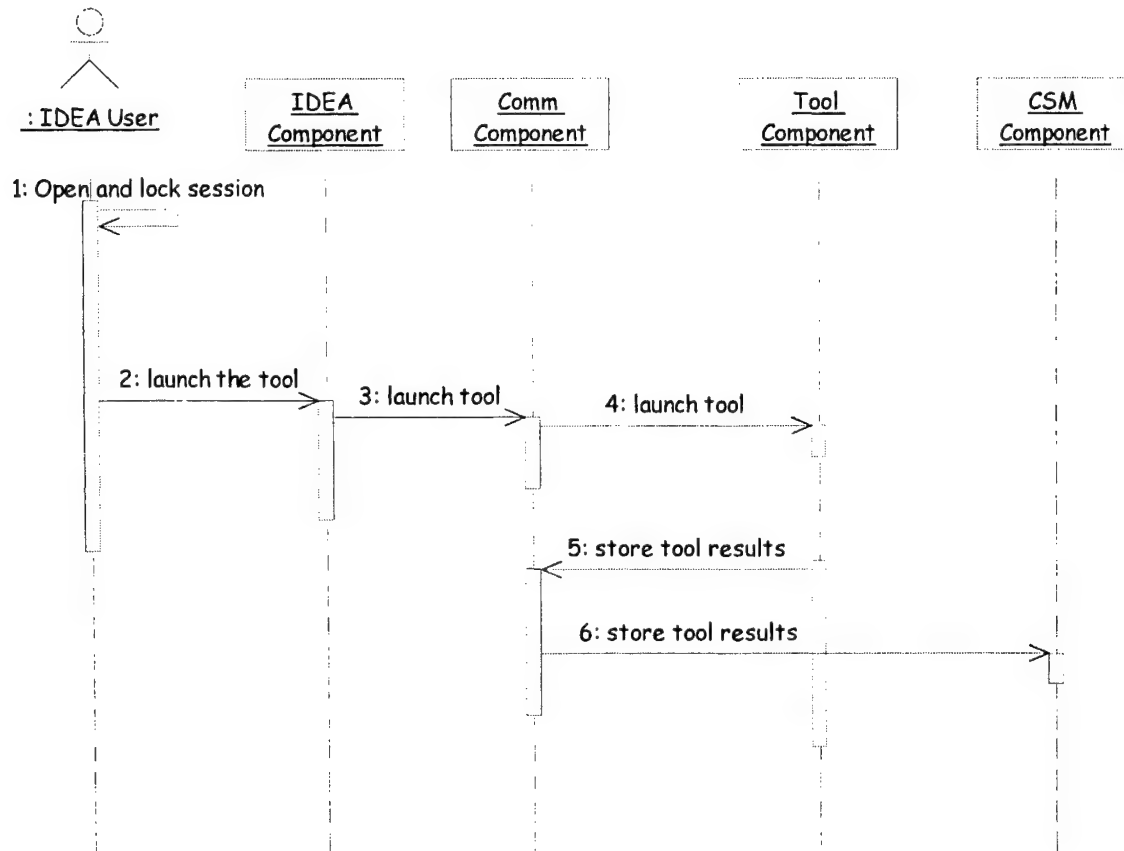


Figure 4-2

4.2 Use Case Launch Discover Tool

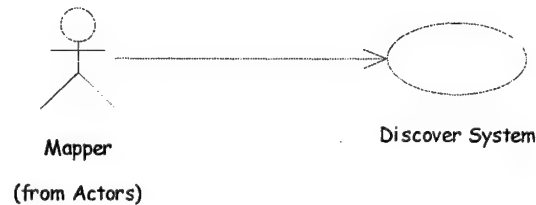


Figure 4-3 - Launch Discover Tool Use Case

4.2.1 Basic Description

Prior to launching scanner and analysis tools, information about the network model for the system must be collected. Discovery tools automatically collect this information.

The user selects a session to run a discover tool on. IDEA opens the session and launches the discover tool. After the tool has finished discovering the system, its information is imported back into the session.

4.2.2 Pre-Conditions

1. One or more discovery tools have been configured and assigned to the session. This pre-condition is tested for and the alternative flow details the behavior if the pre-condition is not met.
2. A session that has not had a discovery tool executed for it exists within the repository.
3. A session has been opened using the Open Session use case.

4.2.3 Flow of Events

4.2.3.1 Start of Use Case

This use case starts when the IDEA User chooses to launch a discover tool.

4.2.3.2 Basic Flow

1. Choose discovery tool

A discovery tool is selected from the session's tool list. A message is sent to the tool manager to launch the selected tool.

2. Launch discovery Tool

The tool object is opened. It gets its configuration information from the repository. After getting its configuration information it calls the COTS tool. The tool executes.

3. Update repository with the session information

After the tool executes, the results are sent to the repository.

4. Format results

The repository component converts the tool's session information format to a format compatible with the database internal session information format. The information is then stored into the database.

4.2.3.3 End of use case

This use case ends when the tool completes and stores its information in the database.

4.2.3.4 Alternative flows

1a. No discovery tools available

In step 1, if the session's discovery tool list is empty, the user is notified and this use case ends.

2a. Discovery tool is not properly configured

In step 2, if the discovery tool is unable to launch the user is notified and this use case ends.

3a. Unable to update session

In step 3, if an error occurs while attempting to update the repository with the additional session information the user is notified and this use case ends.

4a. Unable to format results

In step 4, if the session formatter is unable to format the results, the user is notified and this use case ends.

4.2.4 Post-Conditions

1. If the use case successfully completes, the session model contains information ready to be scanned or analyzed by a scan or analysis tool (respectively).
2. A session is updated.

4.2.5 Special Requirements

Discovery tools are launched within the Tool Component.

4.2.6 Scenarios

4.2.6.1 Scenario 1 - Basic flow

Table 4-1 - Launch Discover Tool Scenario 1

Initiator	Behavior	Target
IDEA Component	Get a discover tool to launch	IDEA Component
IDEA Component	Launch the discover tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the discover tool	External Tool
Tool Component	Store the tool results	Repository Component

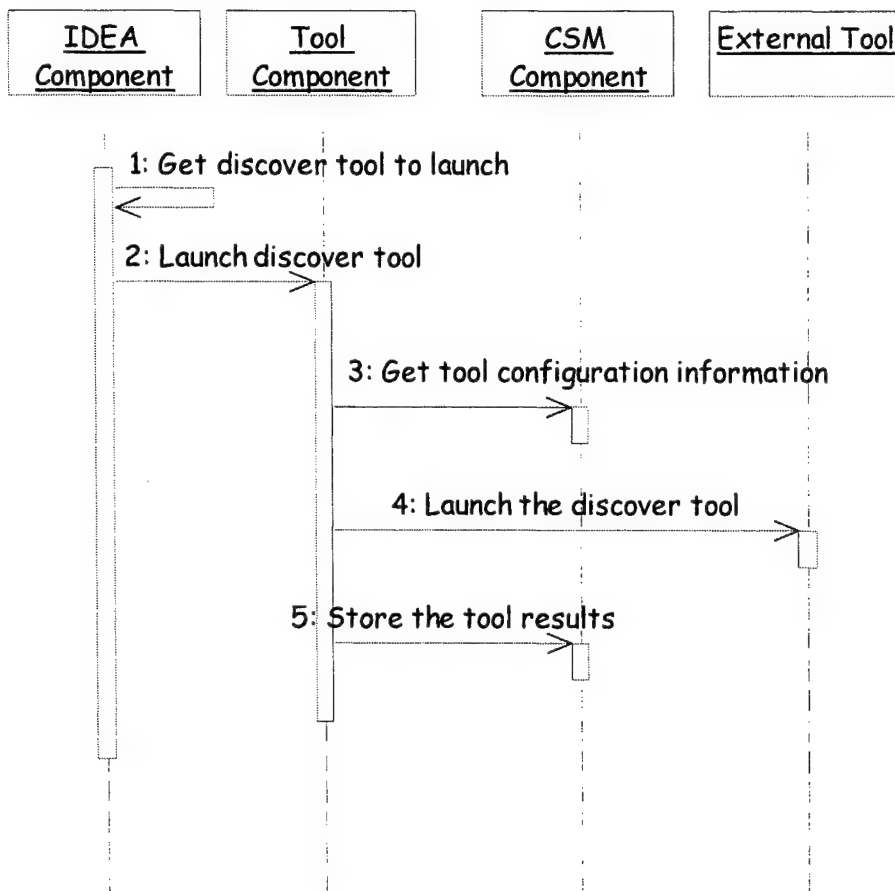


Figure 4-4 - Basic Flow

4.2.6.2 Scenario 2 - No discovery tools available

Table 4-2 - Launch Discover Tool Scenario 2

Initiator	Behavior	Target
IDEA Component	Get a discover tool to launch	IDEA Component
IDEA Component	No tools available	IDEA User

In the scenario selection list, the user has selected a scenario that has no discovery tools assigned to it. When the user tries to launch discovery against the session, this error occurs.

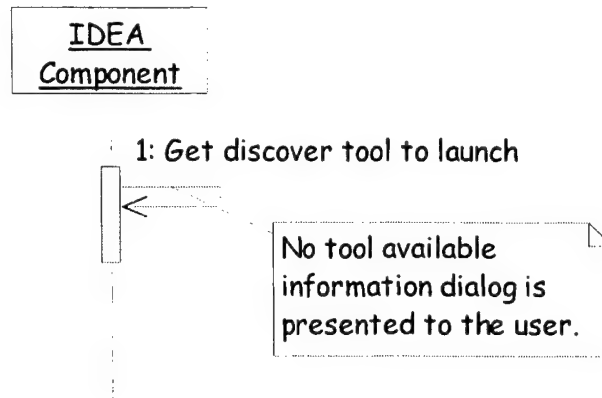


Figure 4-5 - No discovery tools available

4.2.6.3 Scenario 3 - Discovery tool is not properly configured

Table 4-3 - Launch Discover Tool Scenario 3

Initiator	Behavior	Target
IDEA Component	Get a discover tool to launch	IDEA Component
IDEA Component	Launch the discover tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Repository Component	Tool not properly configured	Tool Component
Tool Component	Unable to launch tool	IDEA Component
IDEA Component	Unable to launch tool	IDEA User
IDEA User	Continue with next tool*	IDEA Component

*Or the user can abort the discovery.

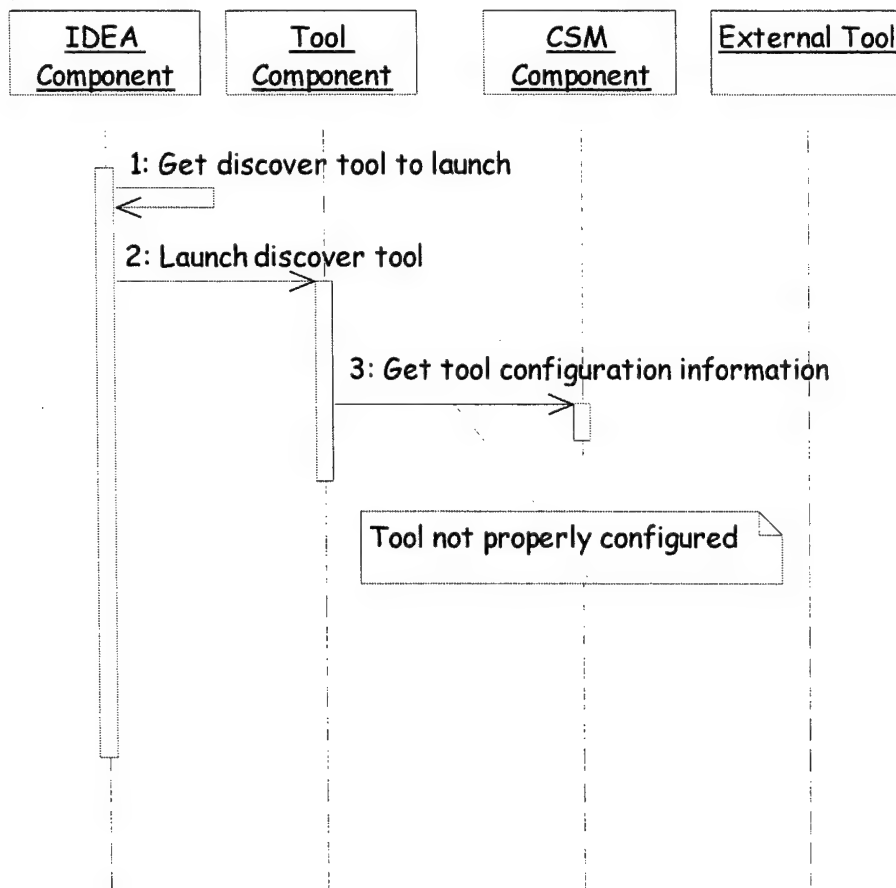


Figure 4-6 - Discovery tool is not properly configured

4.2.6.4 Scenario 4 - Unable to update session

Table 4-4 - Launch Discover Tool Scenario 4

Initiator	Behavior	Target
IDEA Component	Get a discover tool to launch	IDEA Component
IDEA Component	Launch the discover tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the discover tool	External Tool
Tool Component	Store the tool results	Repository Component
Repository Component	Unable to save information	Tool Component
Tool Component	Unable to save results	IDEA Component
IDEA Component	Unable to save results	IDEA User

This differs from Scenario 5 in that the information cannot, for some reason, be stored into the repository.

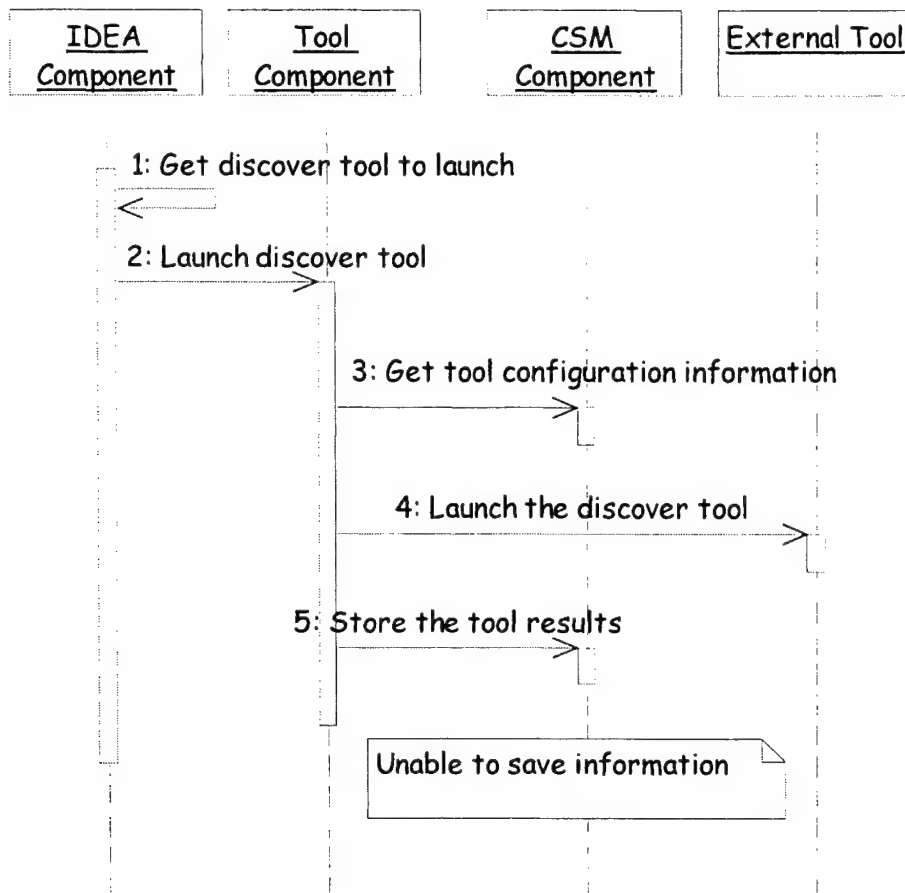


Figure 4-7 - Unable to update session

4.2.6.5 Scenario 5 - Unable to format results

Table 4-5 - Launch Discover Tool Scenario 5

Initiator	Behavior	Target
IDEA Component	Get a discover tool to launch	IDEA Component
IDEA Component	Launch the discover tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the discover tool	External Tool
Tool Component	Store the tool results	Repository Component
Repository Components	Unable to format the results	Tool Component
Tool Component	Tool result version mismatch	IDEA Component
IDEA Component	Tool result version mismatch	IDEA User

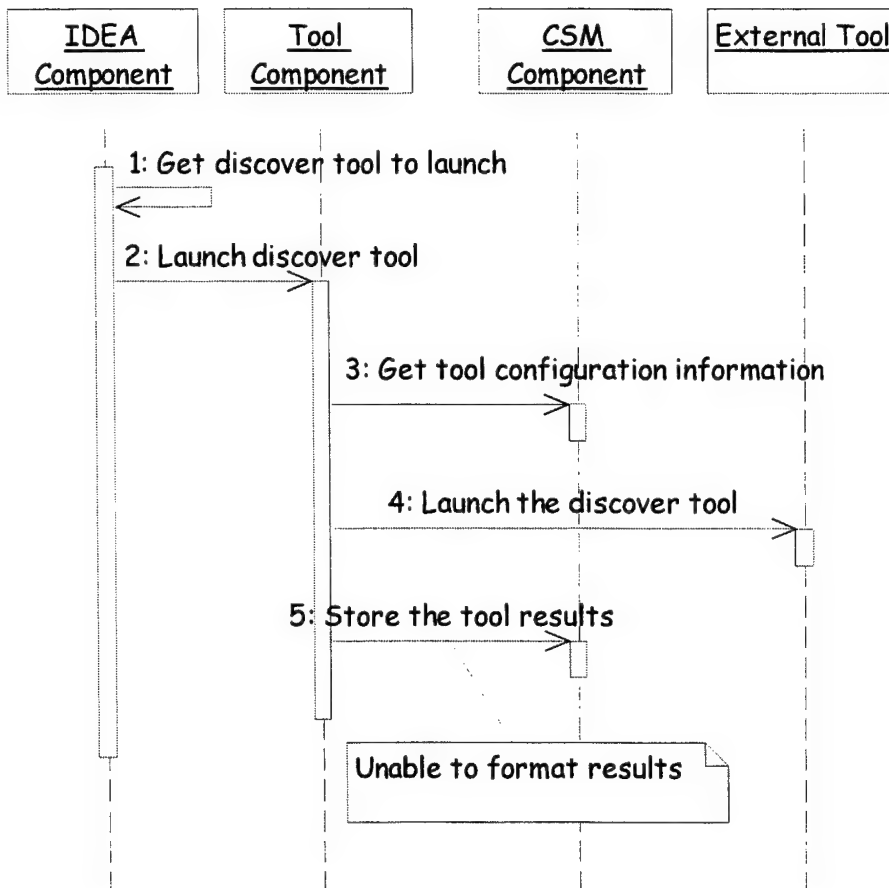


Figure 4-8 - Unable to format results

4.3 Use Case Launch Scan Tool

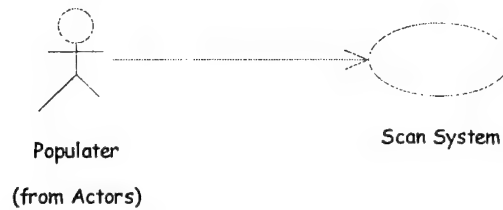


Figure 4-9 - Scan System Use Case

This use case extends Launch Analysis.

4.3.1 Brief Description

Scanning tools are used to gather information for the selected session. A list of scan tools, found in the session information is iterated through.

The session information required by the current scan tool is formatted and passed as the scan tool is launched. Scan results are incorporated back into the session.

4.3.2 Pre-Conditions

1. A session that has had a discovery tool executed for it exists within the repository. The Launch Discover Tool use case has been executed.
2. A session has been opened using the Open Session use case.

4.3.3 Flow of Events

4.3.3.1 Start of use case

This use case starts when the user chooses to launch a scan tool.

4.3.3.2 Basic flow

1. Choose scan tool

A scan tool is selected from the session's tool list. A message is sent to the tool manager to launch the selected tool.

2. Launch scan tool

The tool object is opened. It gets its configuration information from the repository. After getting its configuration information it calls the external tool. The tool executes.

3. Update repository with the session information

After the tool executes, the results are sent to the repository.

4. Format results

The repository component converts the tool's session information format to a format compatible with the database internal session information format. The information is then stored into the database.

4.3.3.3 End of use case

This use case ends when the tool completes and stores its information in the database.

4.3.3.4 Alternative flows

1a. No scan tools available

In step 1, if the session's scan tool list is empty, the user is notified and this use case ends.

2a. Scan tool is not properly configured

In step 2, if the scan tool is unable to launch the user is notified and this use case ends.

3a. Unable to update session

In step 3, if an error occurs while attempting to update the repository with the additional session information the user is notified and this use case ends.

4a. Unable to format results

In step 4, if the session formatter is unable to format the results, the user is notified and this use case ends.

4.3.4 Post-Conditions

1. If the use case successfully completes, the session model contains information ready to be analyzed by an analysis tool.
2. The session is updated.

4.3.5 Special Requirements

4.3.6 Scenarios

4.3.6.1 Analysis

```
open the scanner tool list,  
iterate through the scanner list  
    each scanner knows what session information it requires and how to get it  
    after getting the session information the scanner formats it and launches a  
    tool  
    results of the tool are read and passed back into the repository  
close the scanner tool list.
```

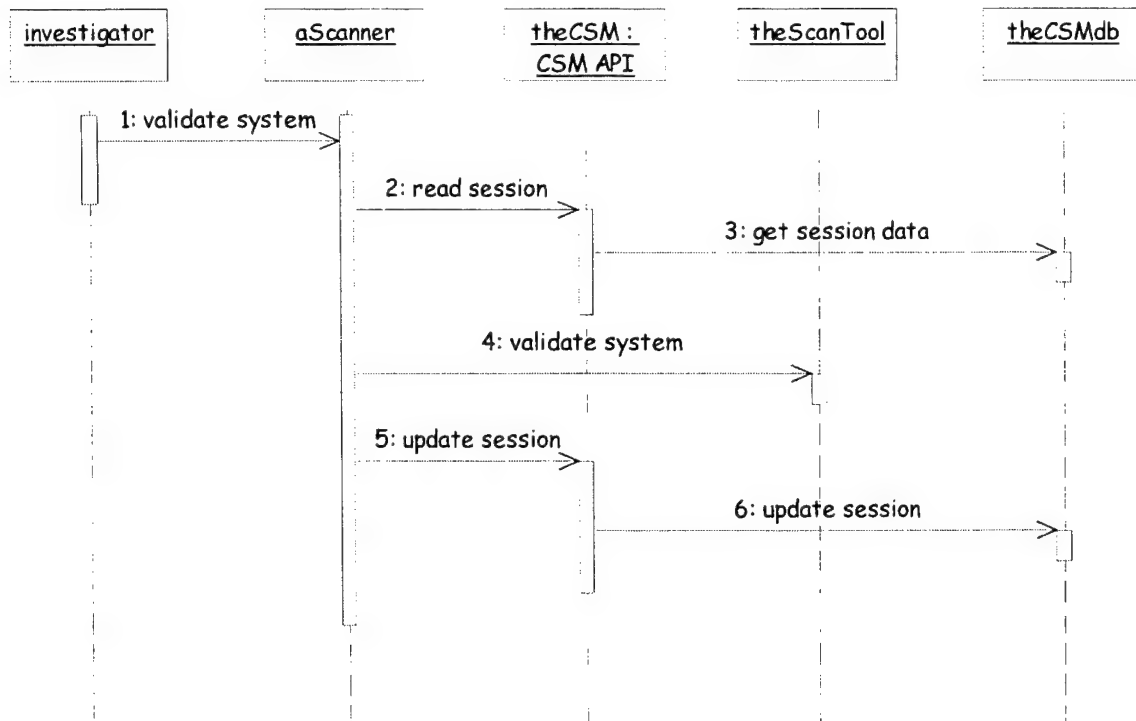



Figure 4-10 - Scan System Analysis Interaction Diagram

4.3.6.2 Preliminary

The interaction diagram for this use case is identical to that of the Launch Analysis use case preliminary sequence diagram. The data passed between components would be the qualifier for the specific scan tool.

4.3.6.3 Scenario 1 - Basic flow

Table 4-6 - Launch Scan Tool Scenario 1

Initiator	Behavior	Target
IDEA Component	Get a scan tool to launch	IDEA Component
IDEA Component	Launch the scan tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the scan tool	External Tool
Tool Component	Store the tool results	Repository Component

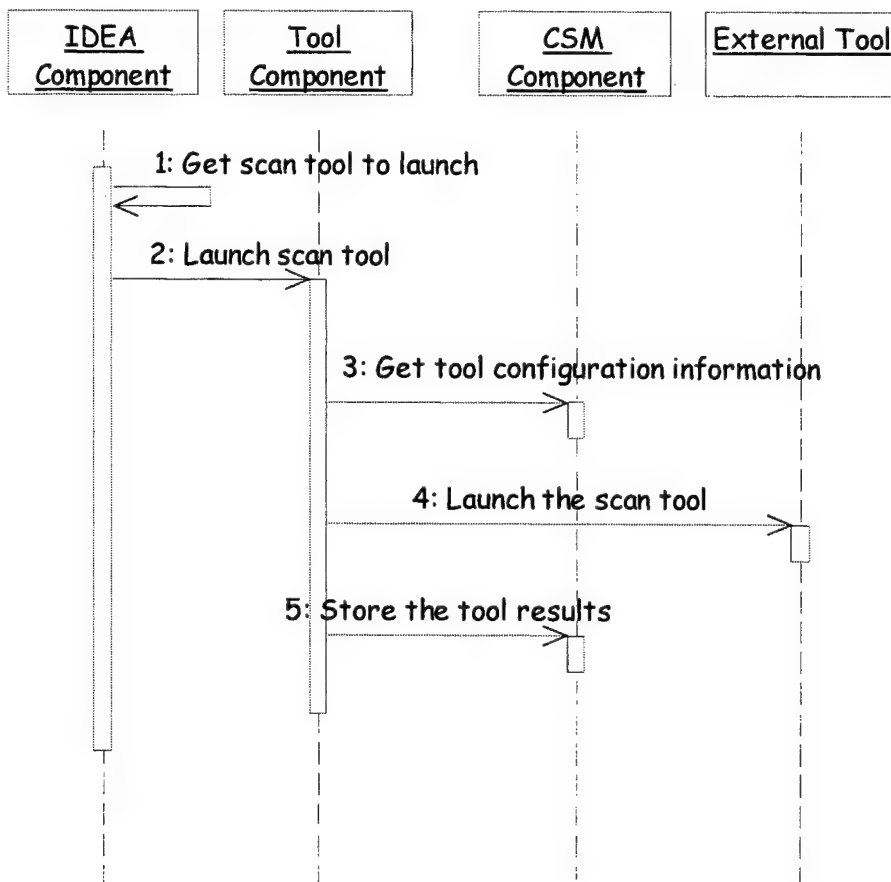


Figure 4-11 - Basic flow

4.3.6.4 Scenario 2 - No scan tools available

Table 4-7 - Launch Scan Tool Scenario 2

Initiator	Behavior	Target
IDEA Component	Get a scan tool to launch	IDEA Component
IDEA Component	Report to User	IDEA User

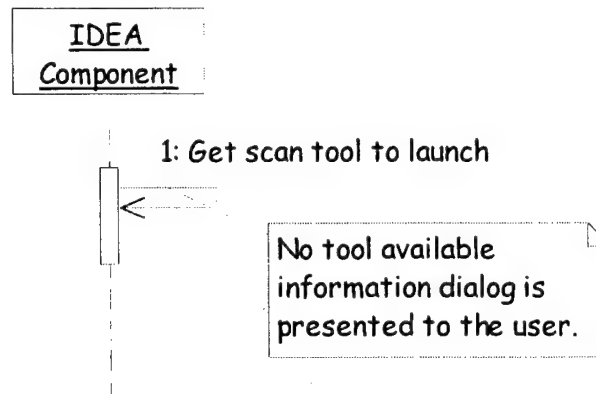


Figure 4-12 - No scan tools available

4.3.6.5 Scenario 3 - Scan tool is not properly configured

Table 4-8 - Launch Scan Tool Scenario 3

Initiator	Behavior	Target
IDEA Component	Get a scan tool to launch	IDEA Component
IDEA Component	Launch the scan tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Repository Component	Tool not properly configured	Tool Component
Tool Component	Unable to launch tool	IDEA Component
IDEA Component	Unable to launch tool	IDEA User
IDEA User	Continue with next tool*	IDEA Component

*Or abort Scanning.

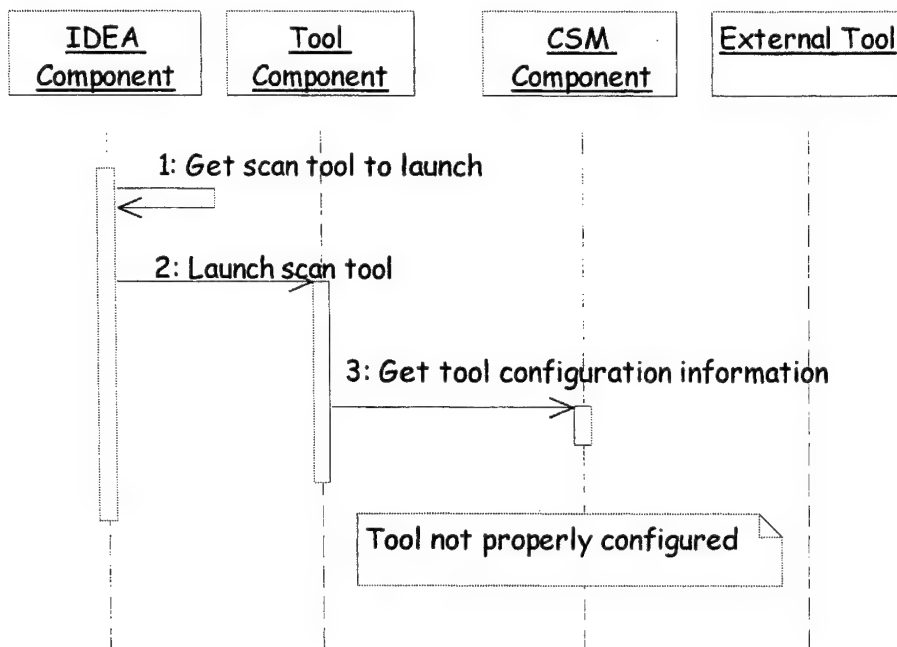


Figure 4-13 - Scan tool is not properly configured

4.3.6.6 Scenario 4 - Unable to update session

Table 4-9 - Launch Scan Tool Scenario 4

Initiator	Behavior	Target
IDEA Component	Get a scan tool to launch	IDEA Component
IDEA Component	Launch the scan tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the scan tool	External Tool
Tool Component	Store the tool results	Repository Component
Repository Component	Unable to save information	Tool Component
Tool Component	Unable to save results	IDEA Component
IDEA Component	Unable to save results	IDEA User

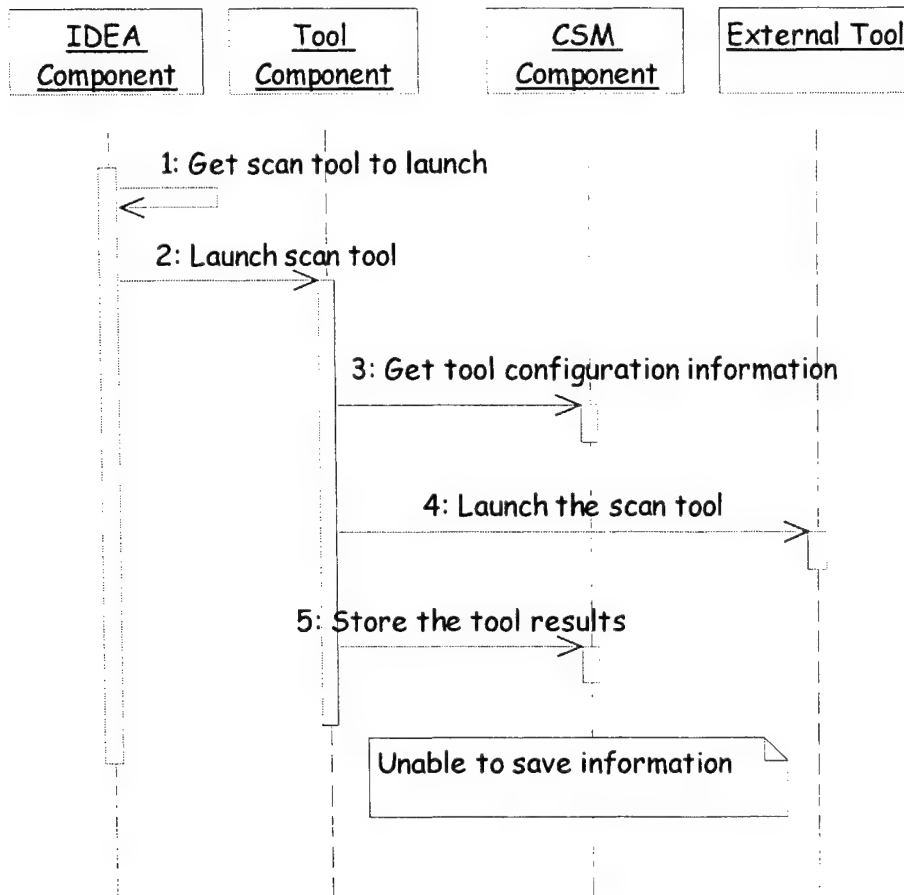


Figure 4-14 - Unable to update session

4.3.6.7 Scenario 5 - Unable to format results

Table 4-10 - Launch Scan Tool Scenario 5

Initiator	Behavior	Target
IDEA Component	Get a scan tool to launch	IDEA Component
IDEA Component	Launch the scan tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the scan tool	External Tool
Tool Component	Store the tool results	Repository Component
Repository Components	Unable to format the results	Tool Component
Tool Component	Tool result version mismatch	IDEA Component
IDEA Component	Tool result version mismatch	IDEA User

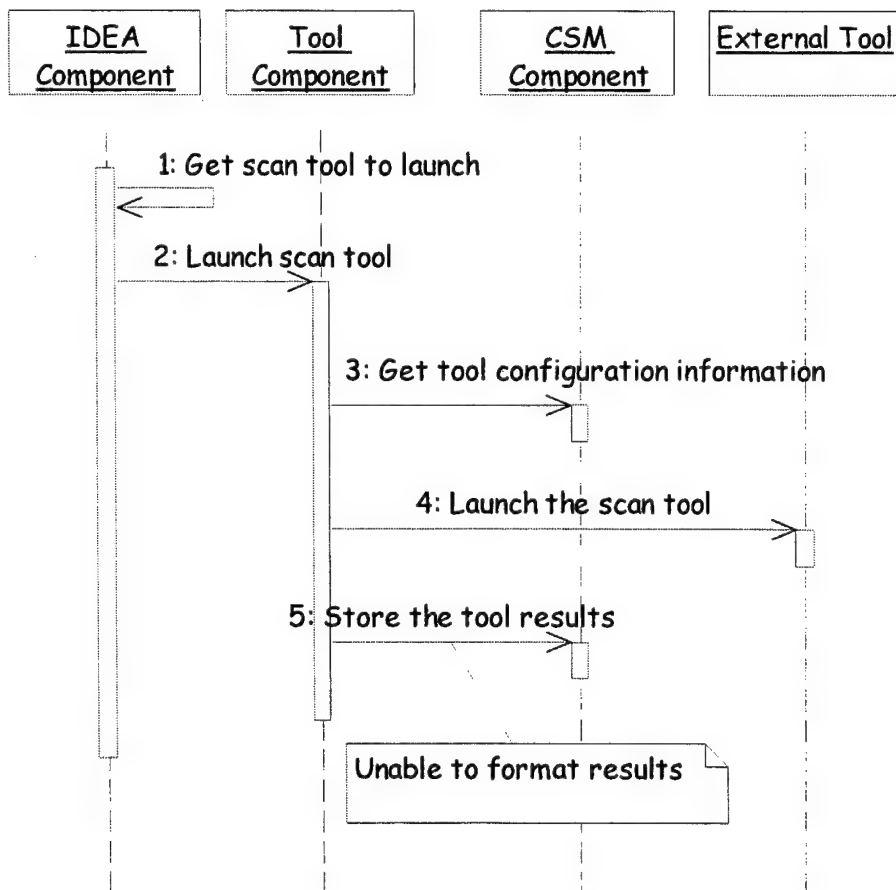


Figure 4-15 - Unable to format results

4.4 Use Case Launch Analysis Tool

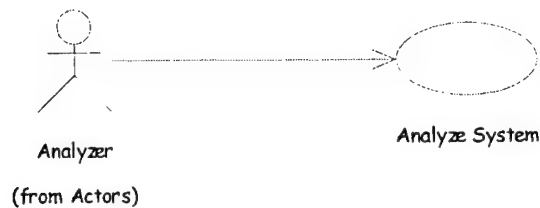


Figure 4-16 - Analyze Model Use Case

This use case extends Launch Analysis.

4.4.1 Brief Description

Analysis tools are used to gather information for the selected session. A list of scan tools, found in the session information is iterated.

The session information required by the current analysis tool is formatted and passed as the analysis tool is launched. Analysis results are incorporated back into the repository.

4.4.2 Pre-Conditions

1. A session that has had a scan tool executed for it exists within the repository. The Launch Scan Tool use case has been executed.
2. A session has been opened using the Open Session use case.

4.4.3 Flow of Events

4.4.3.1 Start of use case

This use case starts when the IDEA User chooses to launch an analysis tool.

4.4.3.2 Basic flow

1. Choose analysis tool

An analysis tool is selected from the session's tool list. A message is sent to the tool manager to launch the selected tool.

2. Launch analysis tool

The tool object is opened. It gets its configuration information from the repository. After getting its configuration information it calls the external tool. The tool executes.

3. Update repository with the session information

After the tool executes, the results are sent to the repository.

4. Format results

The repository component converts the tool's session information format to a format compatible with the database internal session information format. The information is then stored into the database.

4.4.3.3 End of use case

This use case ends when the tool completes and stores its information in the database.

4.4.3.4 Alternative flows

1a. No analysis tools available

In step 1, if the session's analysis tool list is empty, the user is notified and this use case ends.

2a. Analysis tool is not properly configured

In step 2, if the analysis tool is unable to launch the user is notified and this use case ends.

3a. Unable to update session

In step 3, if an error occurs while attempting to update the repository with the additional session information the user is notified and this use case ends.

4a. Unable to format results

In step 4, if the session formatter is unable to format the results, the user is notified and this use case ends.

4.4.4 Post-Conditions

1. If the use case successfully completes, the session model contains information ready to have fuzzy analysis performed on it.
2. The session is updated.

4.4.5 Special Requirements

4.4.6 Scenarios

4.4.6.1 Analysis

```
open the analysis tool list,  
    iterate through the analyzer list  
        each analyzer knows what session information it requires and how to get it  
        after getting the session information the analyzer formats it and launches a  
        tool  
        results of the tool are read and passed back into the repository  
close the analysis tool list.
```

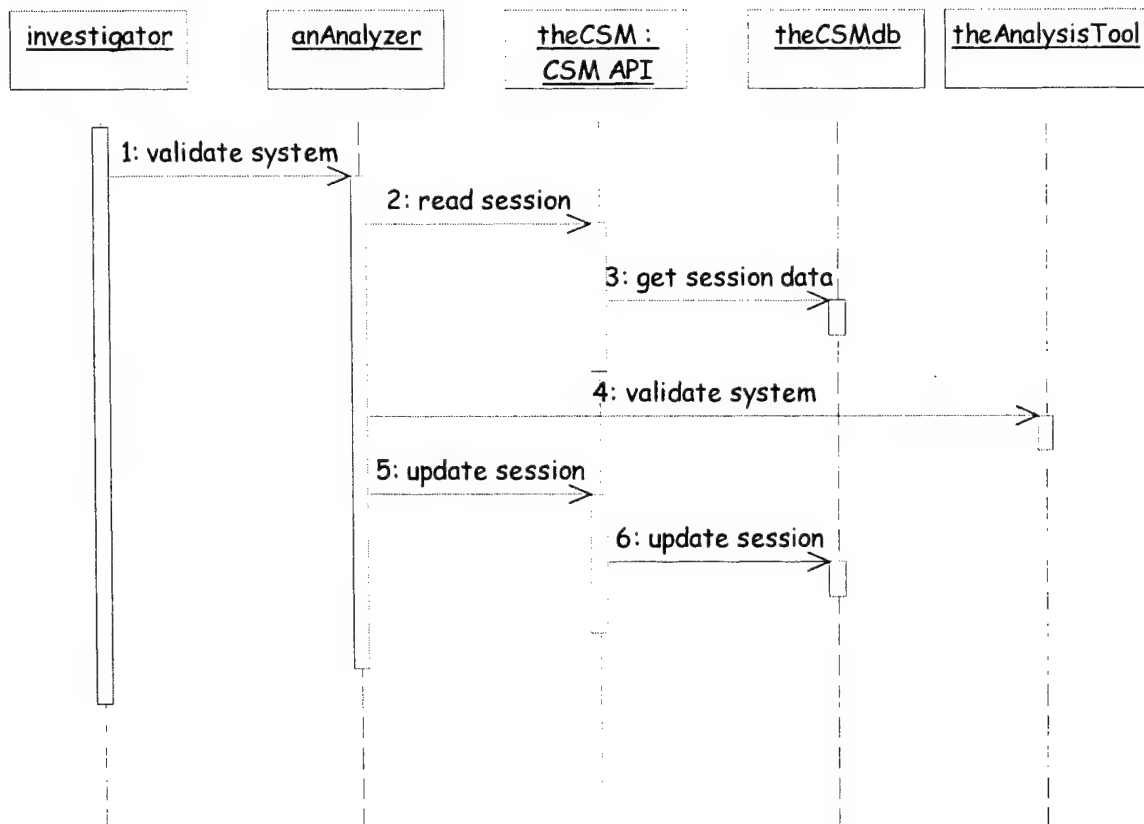



Figure 4-17 - Analyze Model Analysis Interaction Diagram

4.4.6.2 Preliminary

The interaction diagram for this use case is identical to that of the Launch Analysis use case preliminary sequence diagram. The data passed between components would be the qualifier for the specific scan tool.

4.4.6.3 Scenario 1 - Basic flow

Table 4-11 - Launch Analysis Tool Scenario 1

Initiator	Behavior	Target
IDEA Component	Get an analysis tool to launch	IDEA Component
IDEA Component	Launch the analysis tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the analysis tool	External Tool
Tool Component	Store the tool results	Repository Component

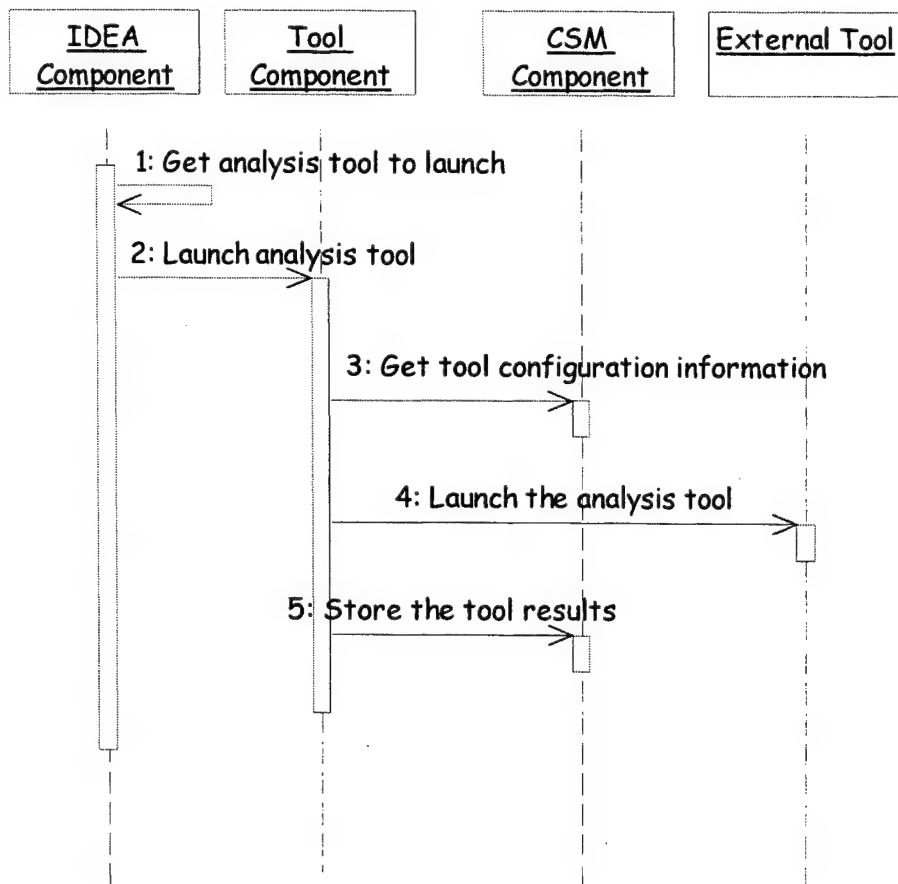


Figure 4-18 - Basic flow

4.4.6.4 Scenario 2 - No Analysis tools available

Table 4-12 - Launch Analysis Tool Scenario 2

Initiator	Behavior	Target
IDEA Component	Get an analysis tool to launch	IDEA Component
IDEA Component	Report to User	IDEA User

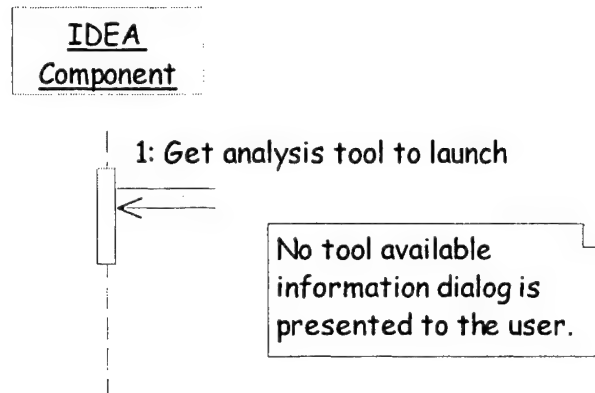


Figure 4-19 - No analysis tool available

4.4.6.5 Scenario 3 - Analysis tool is not properly configured

Table 4-13 - Launch Analysis Tool Scenario 3

Initiator	Behavior	Target
IDEA Component	Get an analysis tool to launch	IDEA Component
IDEA Component	Launch the analysis tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Repository Component	Tool not properly configured	Tool Component
Tool Component	Unable to launch tool	IDEA Component
IDEA Component	Unable to launch tool	IDEA User
IDEA User	Continue with next tool*	IDEA Component

*Or abort Analysis.

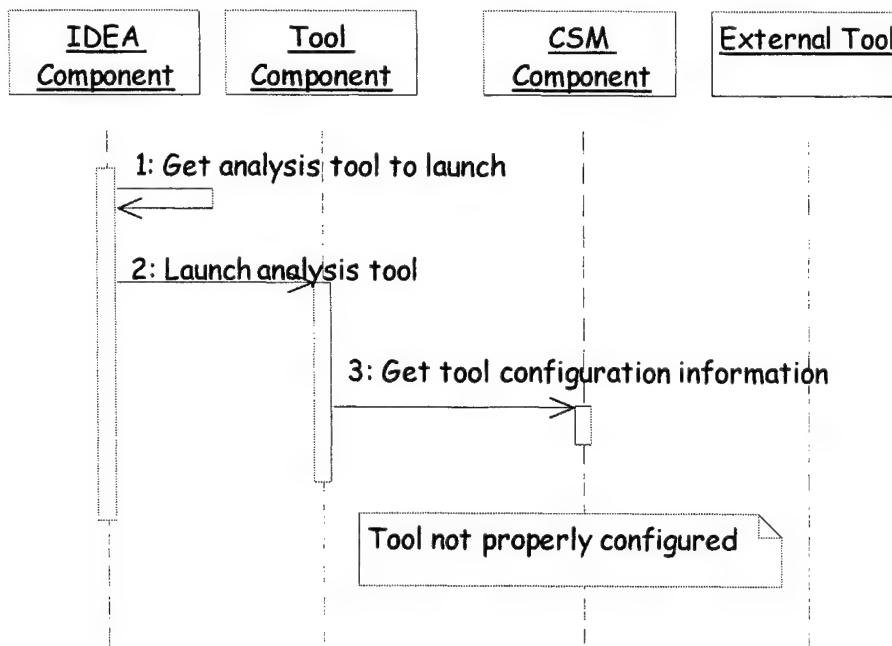


Figure 4-20 - Analysis tool is not properly configured

4.4.6.6 Scenario 4 - Unable to update session

Table 4-14 - Launch Analysis Tool Scenario 4

Initiator	Behavior	Target
IDEA Component	Get an analysis tool to launch	IDEA Component
IDEA Component	Launch the analysis tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the analysis tool	External Tool
Tool Component	Store the tool results	Repository Component
Repository Component	Unable to save information	Tool Component
Tool Component	Unable to save results	IDEA Component
IDEA Component	Unable to save results	IDEA User

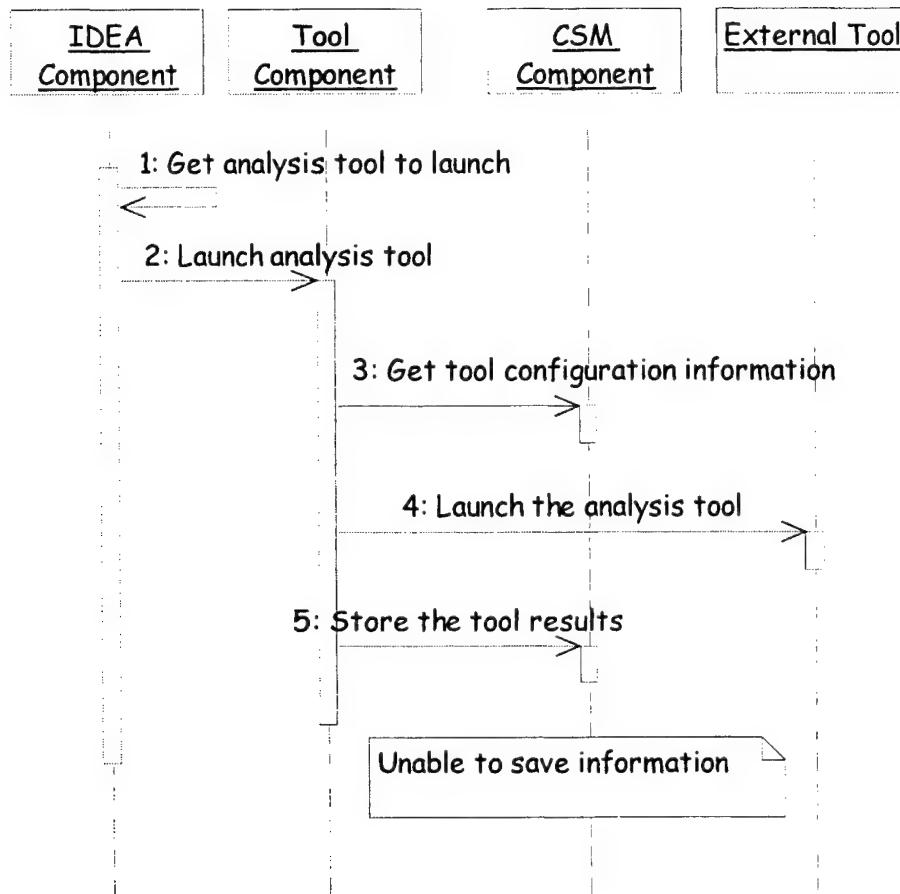


Figure 4-21 - Unable to update session

4.4.6.7 Scenario 5 - Unable to format results

Table 4-15 - Launch Analysis Tool Scenario 5

Initiator	Behavior	Target
IDEA Component	Get an analysis tool to launch	IDEA Component
IDEA Component	Launch the analysis tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the analysis tool	External Tool
Tool Component	Store the tool results	Repository Component
Repository Components	Unable to format the results	Tool Component
Tool Component	Tool result version mismatch	IDEA Component
IDEA Component	Tool result version mismatch	IDEA User

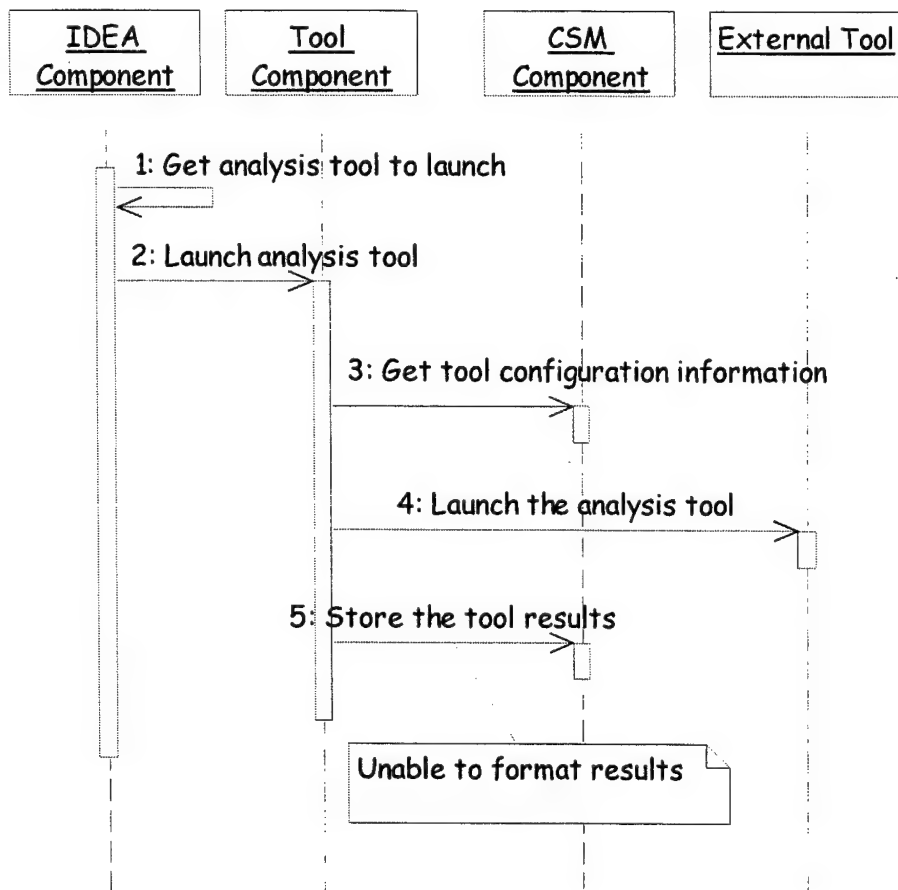


Figure 4-22 - Unable to format results

4.5 Use Case Launch Fuzzy Fusion Analysis

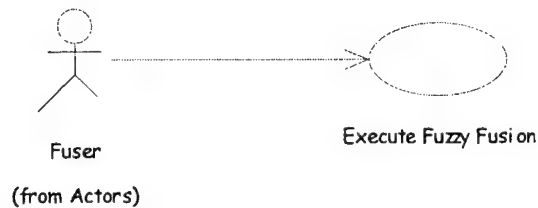


Figure 4-23 - Execute Fuzzy Fusion Use Case

This use case extends Launch Analysis.

4.5.1 Brief Description

Unlike the Scanner and Analysis tools that rely on session for their information, Fuzzy Fusion requires the analysis information with the session information. Fuzzy Fusion explores the results of the scanner and analysis and establishes a vulnerability indicator against the session's data.

4.5.2 Pre-Conditions

3. A session that has had an analysis tool executed for it exists within the repository. The Launch Analysis Tool use case has been executed.
4. A session has been opened using the Open Session use case.

4.5.3 Flow of Events

4.5.3.1 Start of use case

This use case starts when the user chooses to launch the fuzzy analysis tool.

4.5.3.2 Basic flow

1. Choose fuzzy analysis tool

The fuzzy analysis tool is selected from the session's tool list. A message is sent to the tool manager to launch the selected tool.

2. Launch fuzzy analysis tool

The tool object is opened. It gets its configuration information from the repository. After getting its configuration information it calls the external tool. The tool executes.

3. Update repository with the session information

After the tool executes, the results are sent to the repository.

4. Format results

The repository component converts the tool's session information format to a format compatible with the database internal session information format. The information is then stored into the database.

4.5.3.3 End of use case

This use case ends when the tool completes and stores its information in the database.

4.5.3.4 Alternative flows

1a. No fuzzy analysis tool available

In step 1, if the session's fuzzy analysis tool list is empty, the user is notified and this use case ends.

2a. Fuzzy analysis tool is not properly configured

In step 2, if the fuzzy analysis tool is unable to launch the user is notified and this use case ends.

3a. Unable to update session

In step 3, if an error occurs while attempting to update the session with the additional session information the user is notified and this use case ends.

4a. Unable to format results

In step 4, if the session formatter is unable to format the results, the user is notified and this use case ends.

4.5.4 Post-Conditions

3. If the use case successfully completes, the session model contains the updated information.
4. The session is updated.

4.5.5 Special Requirements

4.5.6 Scenarios

4.5.6.1 Analysis

```
open the fuzzy fusion tool,  
    the fuzzy fusion tool knows what session information it requires and how to  
    get it  
    after getting the session information the fuzzy fusion tool formats it and  
    launches FuzzyFusion™.  
    results of the tool are read and passed back into the repository  
close the fuzzy fusion tool.
```

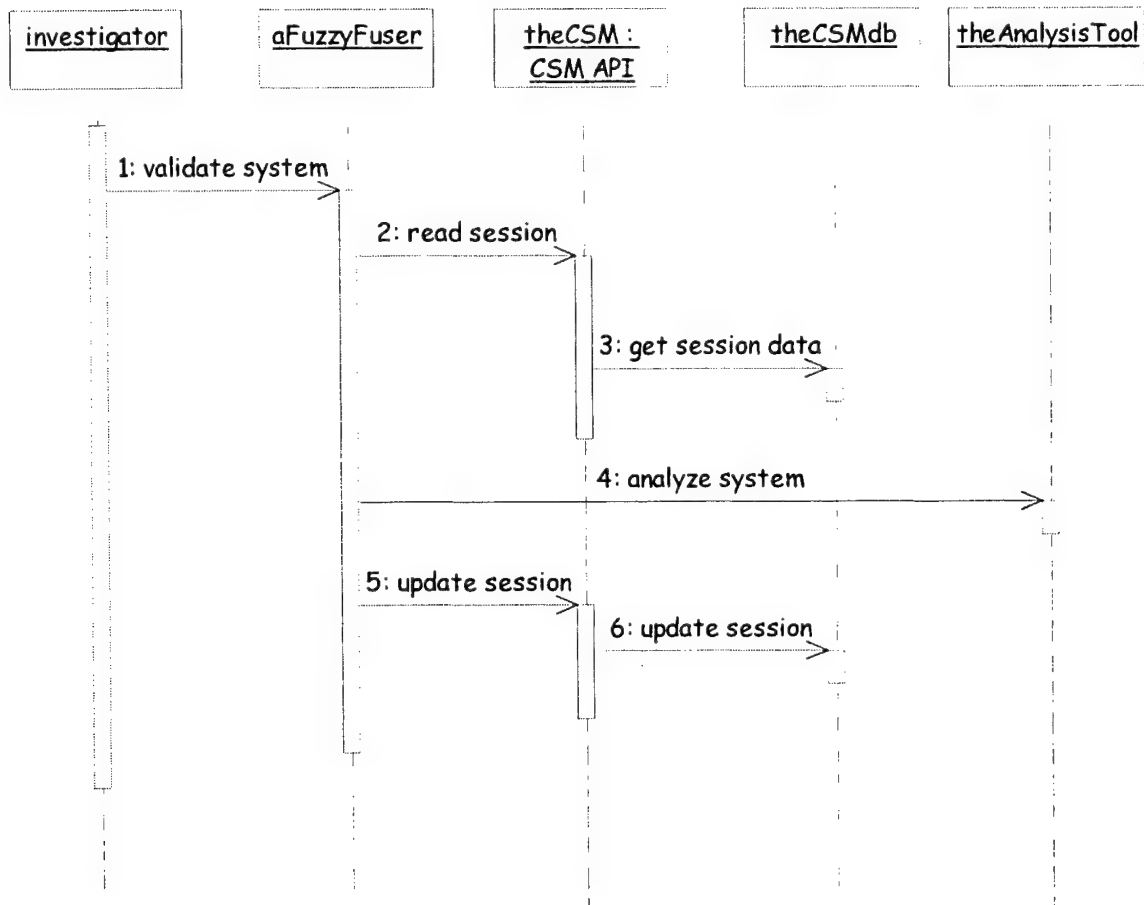



Figure 4-24 - Execute Fuzzy Fusion Analysis Interaction Diagram

4.5.6.2 Preliminary

The interaction diagram for this use case is identical to that of the Launch Analysis use case preliminary sequence diagram. The data passed between components would be the qualifier for the specific scan tool.

4.5.6.3 Scenario 1 - Basic flow

Table 4-16 - Launch Fuzzy Analysis Tool Scenario 1

Initiator	Behavior	Target
IDEA Component	Get a fuzzy analysis tool to launch	IDEA Component
IDEA Component	Launch the fuzzy analysis tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the fuzzy analysis tool	External Tool
Tool Component	Store the tool results	Repository Component

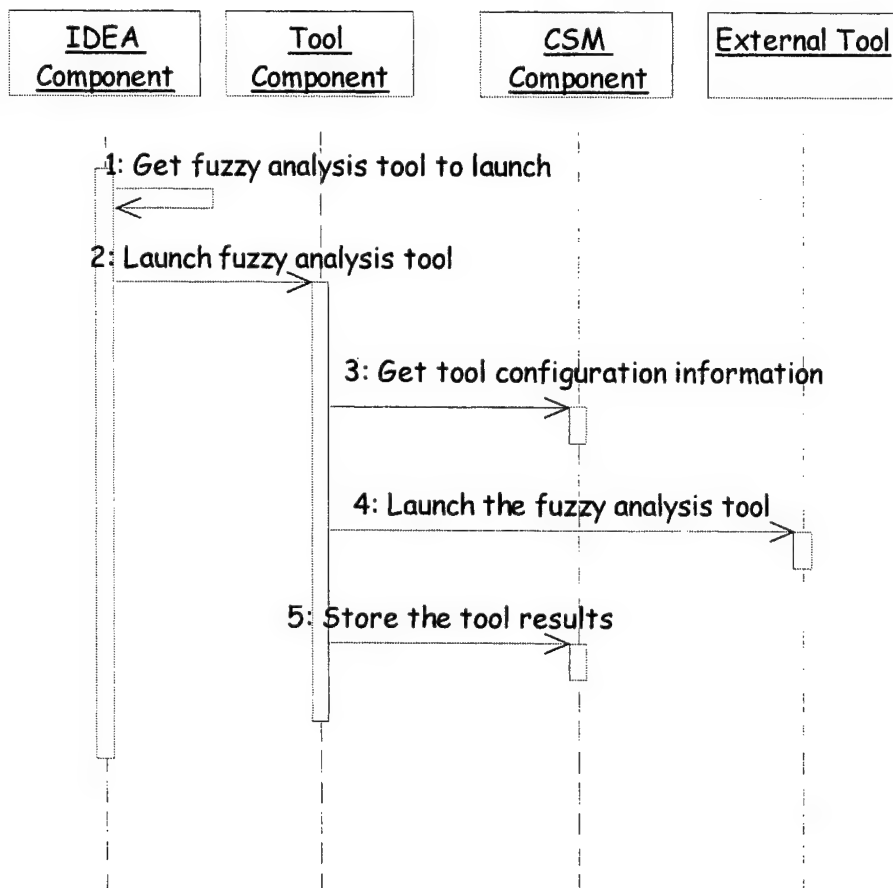


Figure 4-25 - Basic flow

4.5.6.4 Scenario 2 - No fuzzy analysis tool available

Table 4-17 - Launch Fuzzy Analysis Tool Scenario 2

Initiator	Behavior	Target
IDEA Component	Get a fuzzy analysis tool to launch	IDEA Component
IDEA Component	Report to User	IDEA User

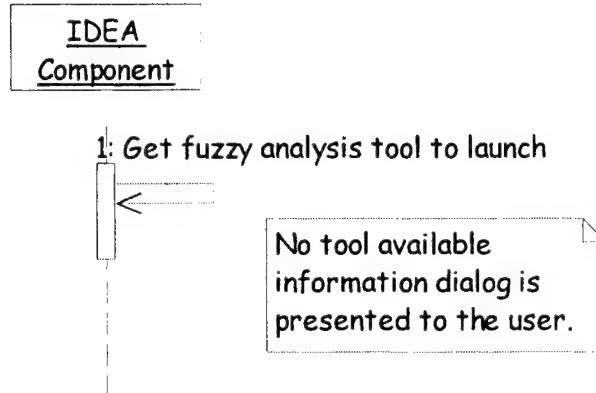


Figure 4-26 - No fuzzy analysis tool available

4.5.6.5 Scenario 3 - Fuzzy analysis tool is not properly configured

Table 4-18 - Launch Fuzzy Analysis Tool Scenario 3

Initiator	Behavior	Target
IDEA Component	Get a fuzzy analysis tool to launch	IDEA Component
IDEA Component	Launch the fuzzy analysis tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Repository Component	Tool not properly configured	Tool Component
Tool Component	Unable to launch tool	IDEA Component
IDEA Component	Unable to launch tool	IDEA User
IDEA User	Continue with next tool*	IDEA Component

*Or abort Analysis.

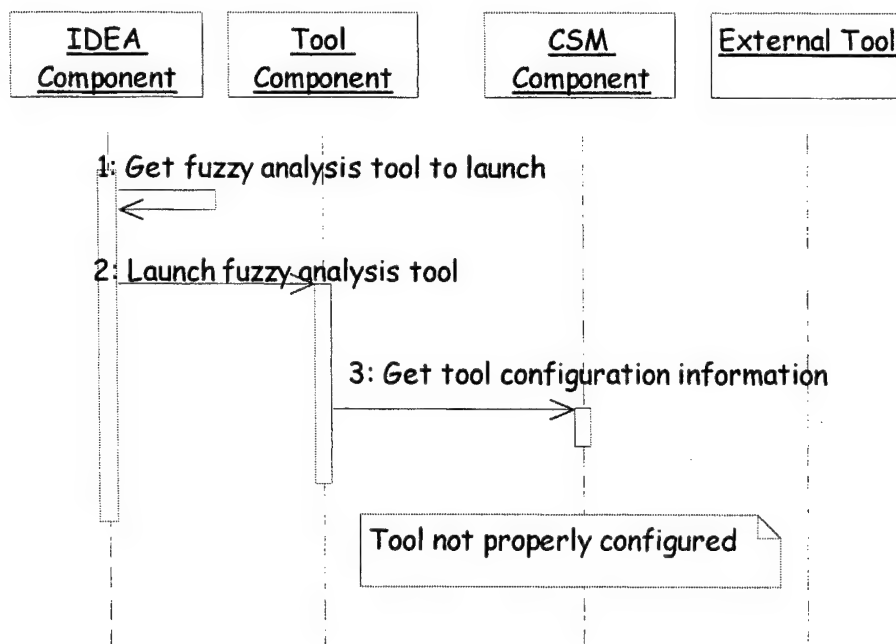


Figure 4-27 - Fuzzy analysis tool is not properly configured

4.5.6.6 Scenario 4 - Unable to update session

Table 4-19 - Launch Fuzzy Analysis Tool Scenario 4

Initiator	Behavior	Target
IDEA Component	Get a fuzzy analysis tool to launch	IDEA Component
IDEA Component	Launch the fuzzy analysis tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the fuzzy analysis tool	External Tool
Tool Component	Store the tool results	Repository Component
Repository Component	Unable to save information	Tool Component
Tool Component	Unable to save results	IDEA Component
IDEA Component	Unable to save results	IDEA User

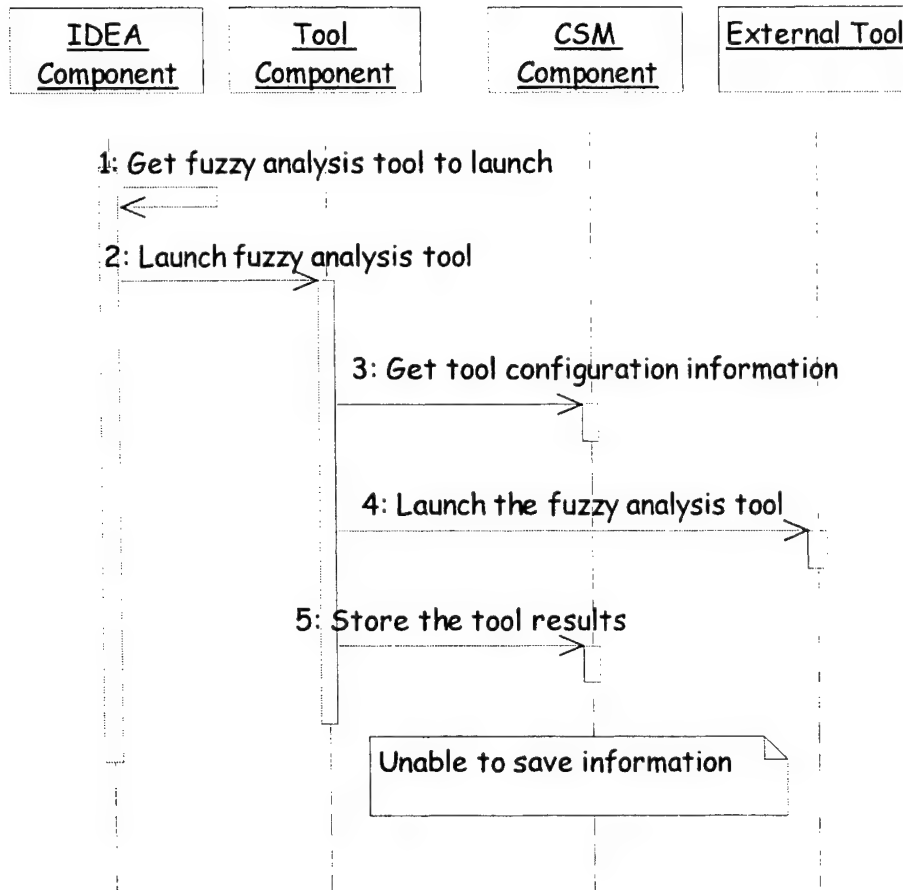


Figure 4-28 - Unable to update session

4.5.6.7 Scenario 5 - Unable to format results

Table 4-20 - Launch Fuzzy Analysis Tool Scenario 5

Initiator	Behavior	Target
IDEA Component	Get a fuzzy analysis tool to launch	IDEA Component
IDEA Component	Launch the fuzzy analysis tool	Tool Component
Tool Component	Get tool configuration information	Repository Component
Tool Component	Launch the fuzzy analysis tool	External Tool
Tool Component	Store the tool results	Repository Component
Repository Components	Unable to format the results	Tool Component
Tool Component	Tool result version mismatch	IDEA Component
IDEA Component	Tool result version mismatch	IDEA User

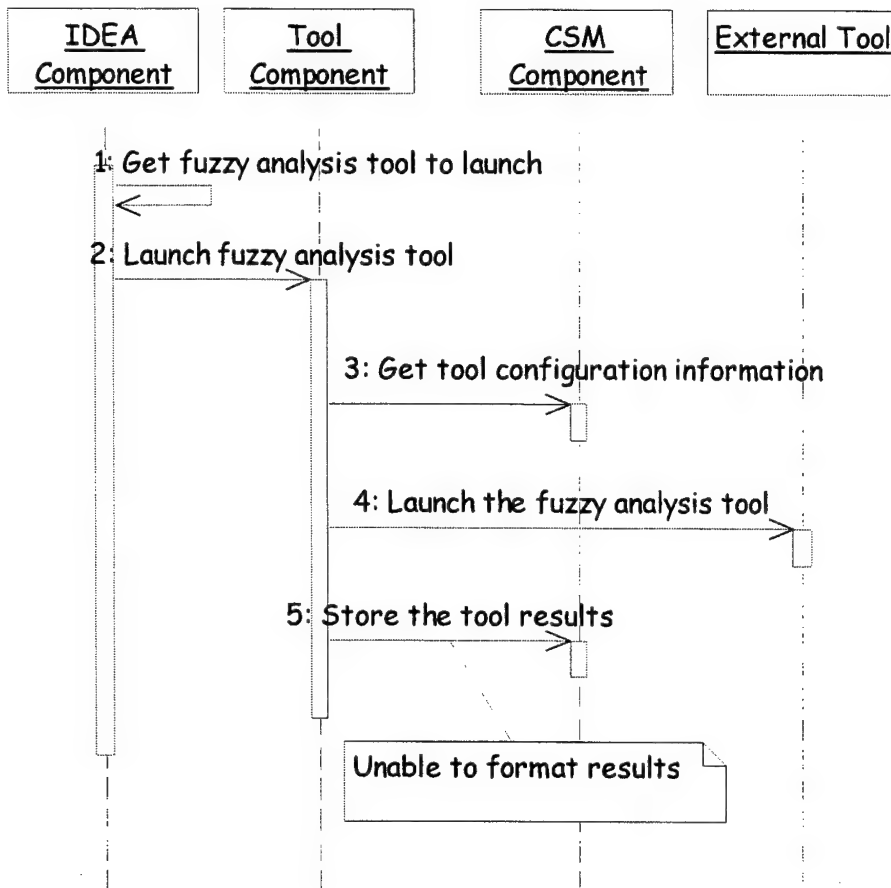


Figure 4-29 - Unable to format results

4.6 Use Case View Results



Figure 4-30 - View Results Use Case

4.6.1 Brief Description

After running an investigation users have the capability to view the results of the analysis. IDEA allows the user to select the session whose results are to be viewed. IDEA then launches a result-browser to view the results report. Users have the option to save the report to file and print it.

4.6.2 Pre-Conditions

The Launch Tool use case has completed.

4.6.3 Flow of Events

4.6.3.1 Start of use case

This use case starts when the user wants to view a session's information.

4.6.3.2 Basic flow

1. Get Session List and Open Session

These use cases are included.

2. Show Session

The session information is presented to the user. The user is given a set of options for the information: Print, Save, Close.

3. Print or Save session

The session information is directed to the printer or a file.

4.6.3.3 End of use case

This use case ends when the user selects Close.

4.6.3.4 Alternative flows

3a. Unable to Print

In step 3, if a printer is not configured, the user is informed and the step ends.

3b. Unable to Save

In step 3, if the save operation is unable to successfully complete, the user is informed and the step ends.

4.6.4 Post-Conditions

The session's information has been shown to the user. The session is closed.

4.6.5 Special Requirements

4.6.6 Scenario Diagrams

4.6.6.1 Preliminary

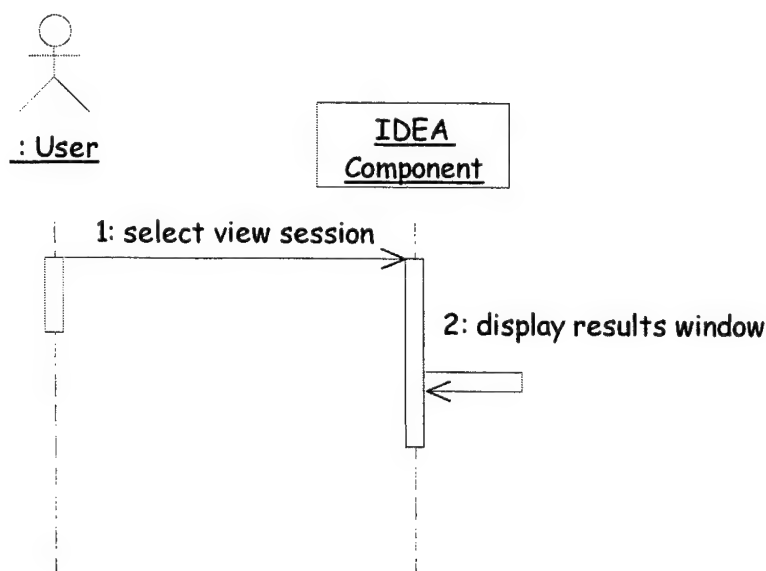


Figure 4-31 - View Results Preliminary Interaction Diagram

This use case executes in part with the Select Analysis Session use case. After the Select Analysis Session use case is performed, this use case continues by displaying a select-to-view-dialog or a method for allowing the user to view the session's tool's results.

After the user opts to view the results for a tool (or a set of tools), the results are displayed to the user. The user also has the option, to save the results to a file or print the results.

4.6.6.2 Scenario 1 - Basic flow

Table 4-21 - View Results Scenario 1

Initiator	Behavior	Target
IDEA User	UC Get Session List	
IDEA User	UC Open Session (as read only)	
IDEA User	Choose to view the session result information	IDEA Component
IDEA Component	Query the database for the session	Repository Component

	information	
IDEA Component	Make the session information human readable	IDEA Component
IDEA Component	Place the session information in a window for the user	IDEA Component
IDEA User	Select Print or Save of the session information	IDEA Component
IDEA Component	Process the user's action	IDEA Component
IDEA User	Close the session information window	IDEA Component

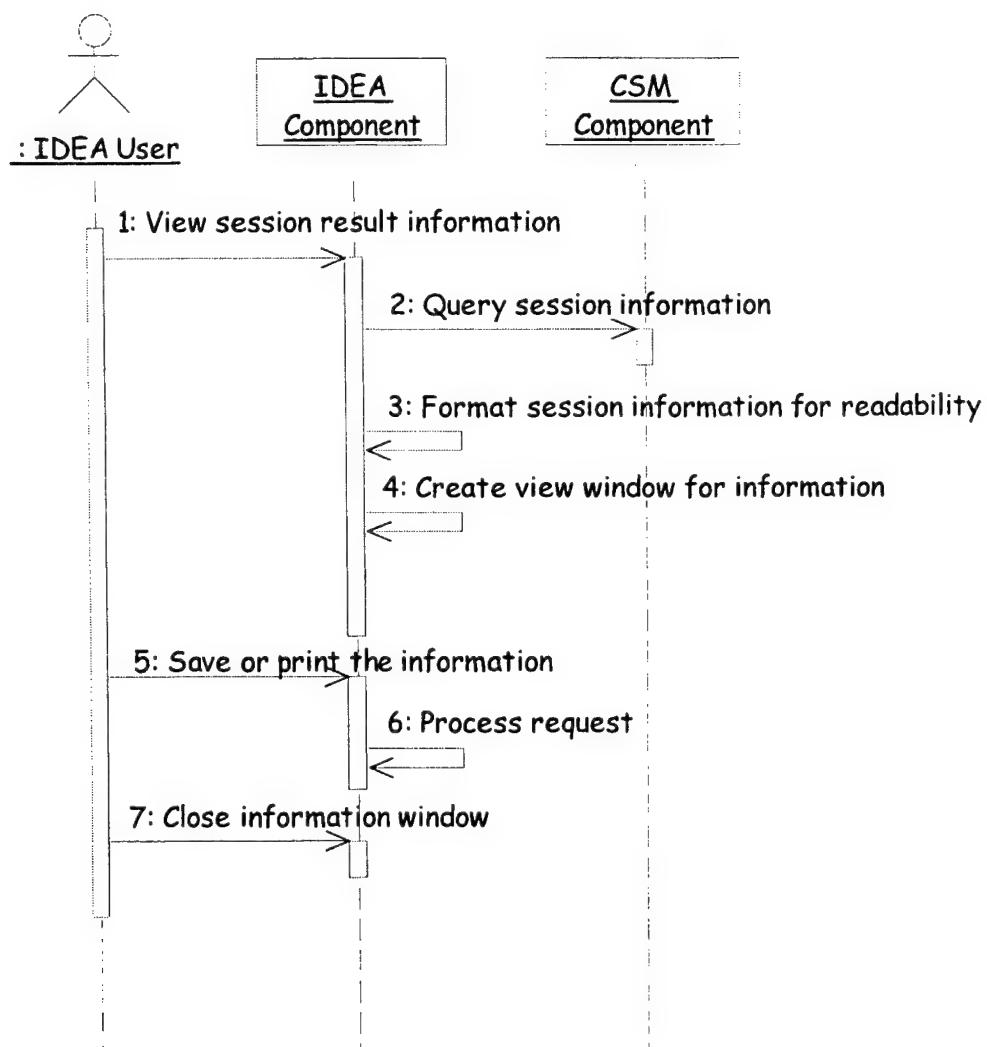


Figure 4-32 - Basic flow

4.6.6.3 Scenario 2 - Unable to print

Table 4-22 - View Results Scenario 2

Initiator	Behavior	Target
IDEA User	UC Get Session List	
IDEA User	UC Open Session (as read only)	
IDEA User	Choose to view the session result information	IDEA Component
IDEA Component	Query the database for the session information	Repository Component
IDEA Component	Make the session information human readable	IDEA Component
IDEA Component	Place the session information in a window for the user	IDEA Component
IDEA User	Select Print or Save of the session information	IDEA Component
IDEA Component	Process the user's print action	IDEA Component
IDEA Component	Inform the user of inability to print	IDEA User

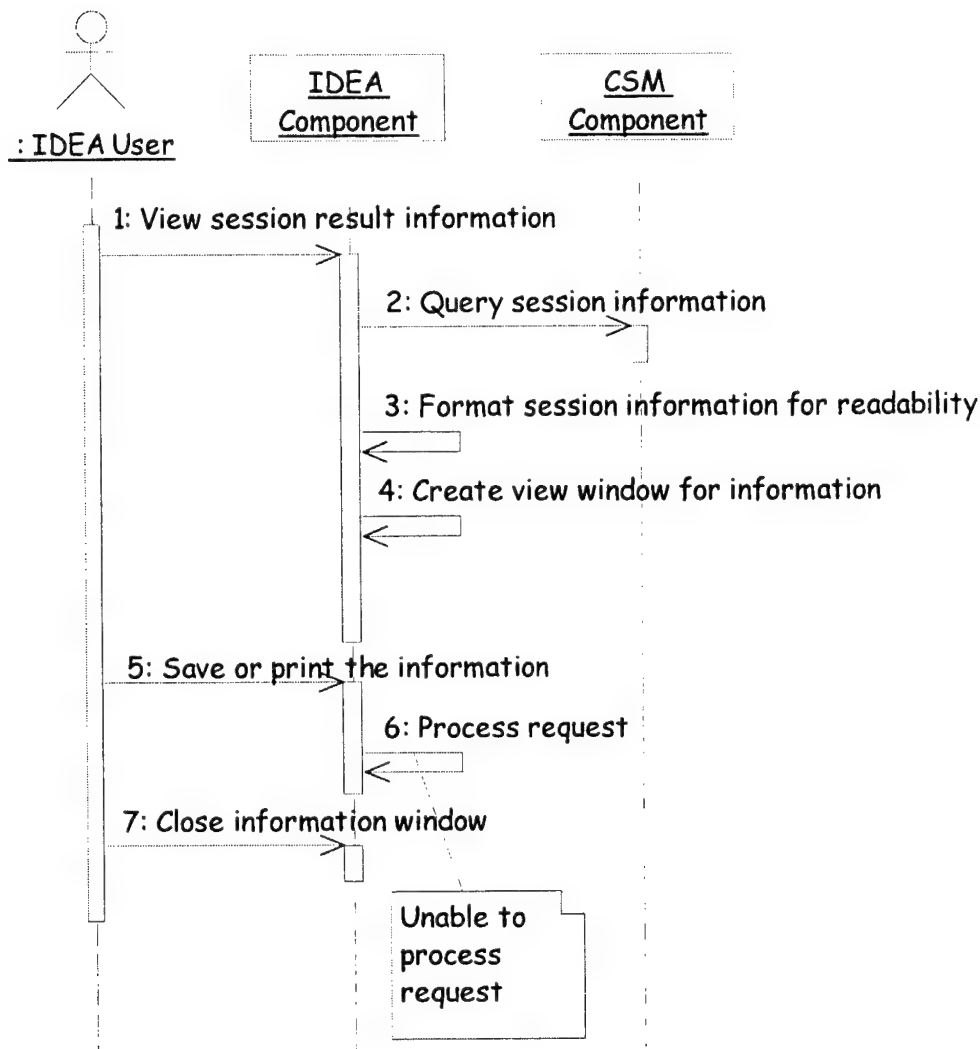


Figure 4-33 - Unable to print

4.6.6.4 Scenario 3 - Unable to save

Table 4-23 - View Results Scenario 3

Initiator	Behavior	Target
IDEA User	UC Get Session List	
IDEA User	UC Open Session (as read only)	
IDEA User	Choose to view the session result information	IDEA Component
IDEA Component	Query the database for the session information	Repository Component

IDEA Component	Make the session information human readable	IDEA Component
IDEA Component	Place the session information in a window for the user	IDEA Component
IDEA User	Select Print or Save of the session information	IDEA Component
IDEA Component	Process the user's save action	IDEA Component
IDEA Component	Inform the user of inability to save	IDEA User

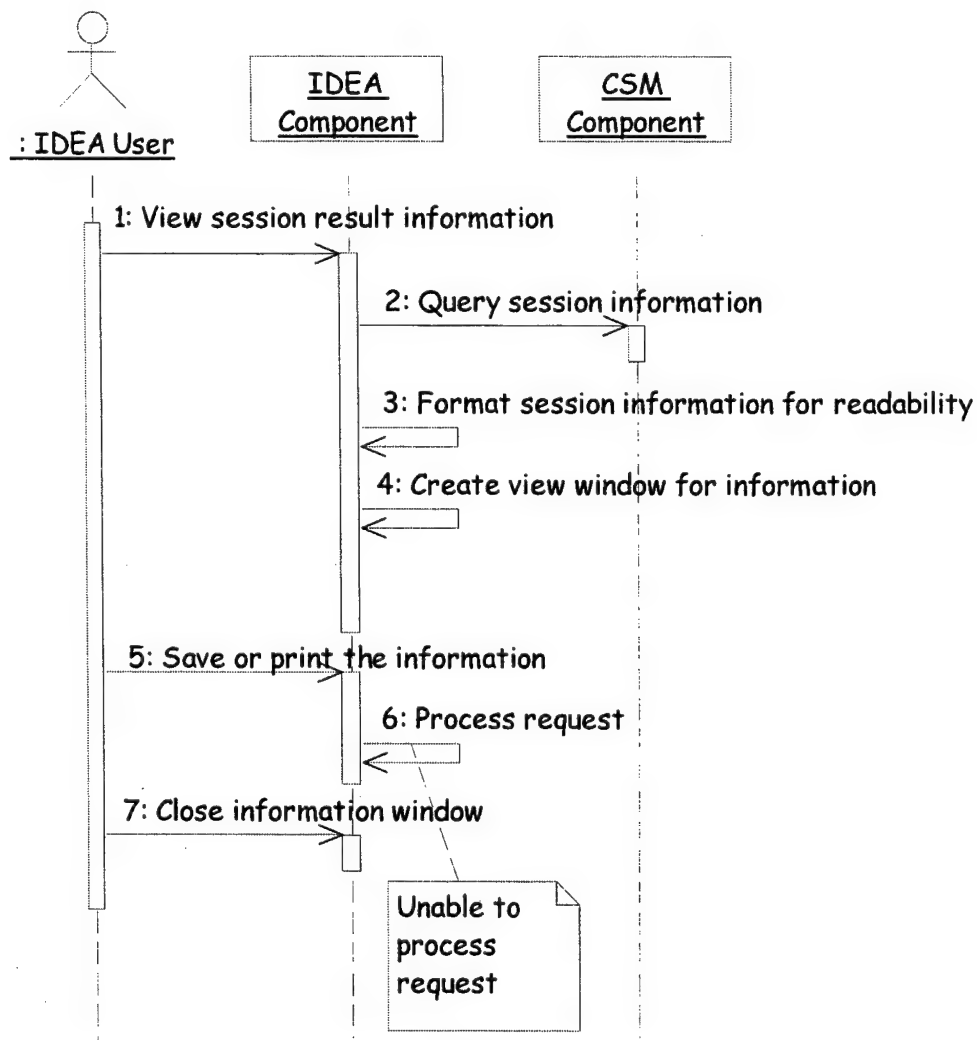


Figure 4-34 - Unable to save

5 Repository Model Use Cases

This section describes various operations for CSM layer design.

A CSM model is composed of well-defined layers, nodes, and node attributes.

Layers represent a part of the physical makeup of a system. Layers are composed of nodes. For example, a Physical-type layer may be composed of PC-type nodes. The individual nodes may connect to nodes (e.g. Hub-type nodes) existing on other layers as well as nodes existing on the same layer. Layers are not connected to other layers.

Nodes represent similar items that compose layers. Node attributes represent the state of the nodes.

The data-model used to represent layers, nodes, and node attributes is shown in Figure Eight.

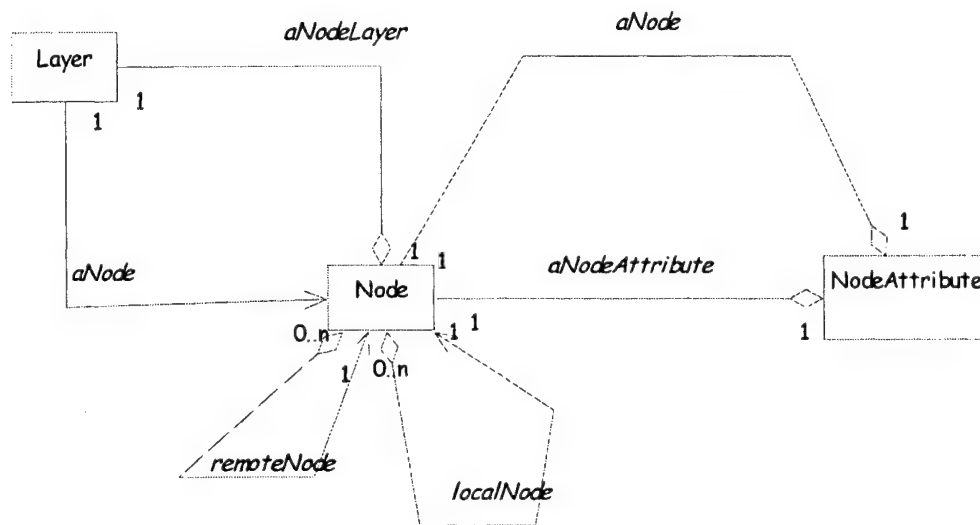


Figure 5-1 - Data Model Diagram

This section presents use cases for defining layers, nodes, attributes, as well as use cases for modifying the model.

5.1 Use Case Get Session List



Figure 5-2 Get Session Lists Use Case

5.1.1 Brief Description

This use case describes how a user gets a list of sessions from the repository.

5.1.2 Pre-Conditions

The CSM has been installed and configured.

Sessions exist within the repository.

5.1.3 Flow of Events

5.1.3.1 Start of use case

This use case starts when it is necessary to retrieve a list of sessions from the repository.

5.1.3.2 Basic Flow

1. Query for list

Query the database for a list of available (unlocked) sessions.

2. Select list

The repository component selects the sessions from the database.

5.1.3.3 End of use case

This use case ends when the user is provided the list of sessions.

5.1.3.4 Alternative Flows

5.1.4 Post-Conditions

5.1.5 Scenario Diagrams

5.1.5.1 Scenario 1 - Basic flow

Table 5-1 - Get Session List Scenario 1

Initiator	Behavior	Target
IDEA User	Tell the IDEA Component to get a list of sessions (all sessions, sessions ready for discovery, scanning, analysis, or fuzzy analysis)	IDEA Component
IDEA Component	Get a list of sessions from the repository	Repository Component
IDEA Component	Display the session to the user	IDEA User

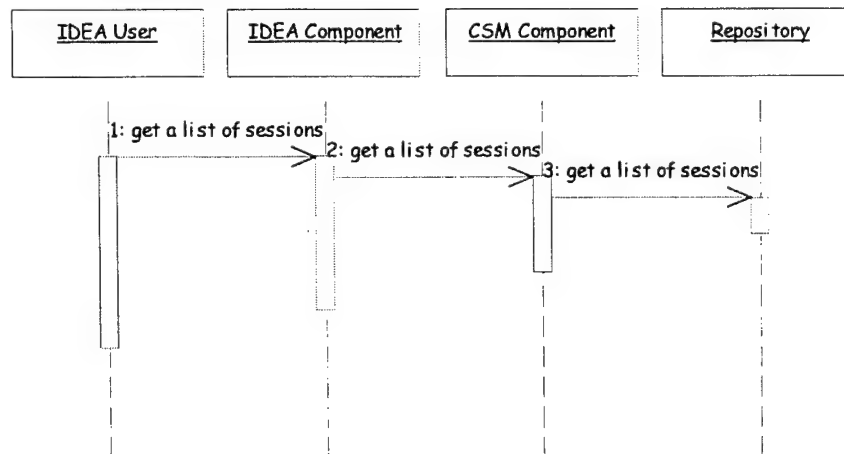


Figure 5-3 - Basic Flow

5.1.5.2 Scenario 2 - No sessions available

Table 5-2 - Get Session List Scenario 2

Initiator	Behavior	Target
IDEA User	Tell the IDEA Component to get a list of sessions (all sessions, sessions ready for discovery, scanning, analysis, or fuzzy analysis)	IDEA Component

IDEA Component	Get a list of sessions from the repository	Repository Component
Repository Component	No sessions available	IDEA Component
IDEA Component	No sessions available	IDEA User

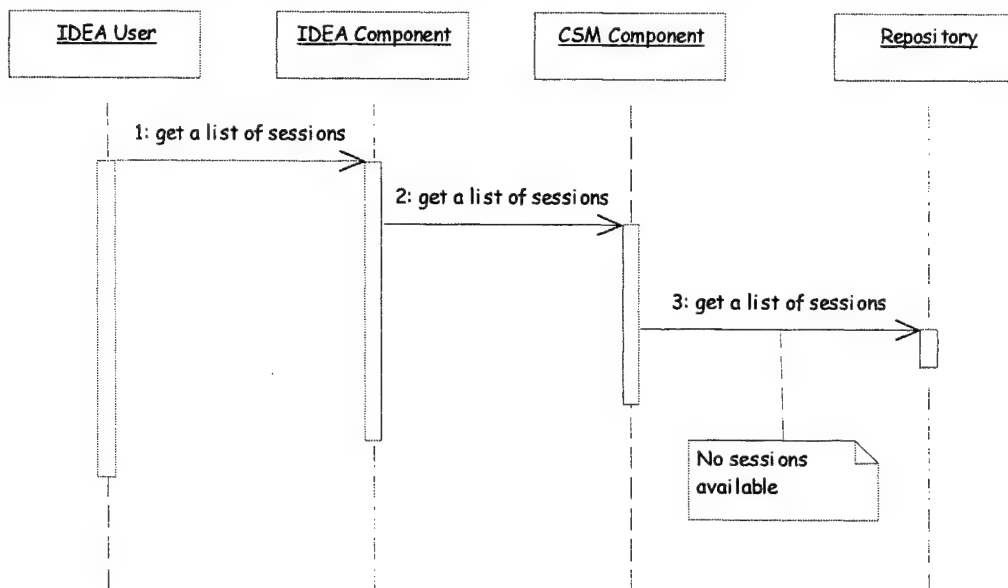


Figure 5-4 - No sessions available

In step 1, the Repository component reports that there are no sessions available. The user is informed that no sessions are available (as opposed to no sessions exist), and the use case ends.

There are two reasons why sessions may not be available: 1) there are no sessions in the database, 2) the user is trying to open a session for write, and all available sessions are already open for writing.

5.1.5.3 Scenario 3 - Unable to find database

Table 5-3 - Get Session List Scenario 3

Initiator	Behavior	Target
IDEA User	Tell the IDEA Component to get a list of sessions (all sessions, sessions ready for discovery, scanning, analysis, or fuzzy analysis)	IDEA Component
IDEA Component	Get a list of sessions from the repository	Repository Component
Repository Component	Database error	IDEA Component
IDEA Component	Database error	IDEA User

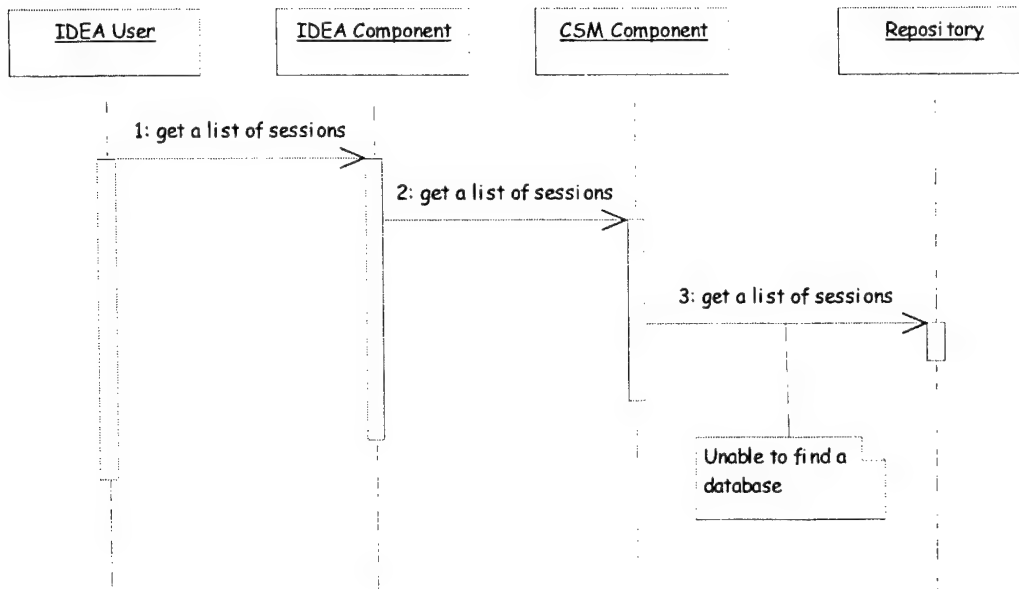


Figure 5-5 - Unable to find a database

In step 2, the Repository component reports that it cannot connect to the database. The user is informed of the existence of a database error and the use case ends.

5.2 Use Case Open Session

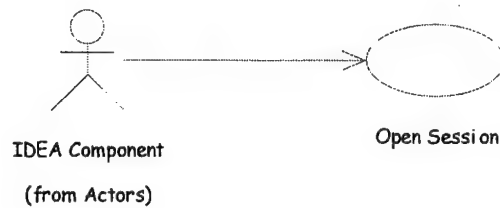


Figure 5-6 - Open Session

5.2.1 Brief Description

In this use case, the IDEA system allows users to run tools against a session.

5.2.2 Pre-Conditions

The session exists and has been properly configured as described in the TBI section.

The session must have been created in the database.

5.2.3 Flow of Events

5.2.3.1 Start of use case

This use case starts when the IDEA User selects the GUI options necessary to map, populate, scan, analyze or fuse a session's information.

5.2.3.2 Basic Flow

1. Get the sessions

The IDEA component is directed, by IDEA User interaction, to get the list of sessions available for discovery, scanning, analysis, or fuzzification. The use case Get Session List shows the flow of this step.

2. Query the database for the session

A query to the database is made to retrieve the specified session with the available tools for the context selected in step 1.

- If the context is discovery a list of discovery tools for the session is returned.
- If the context is scan, a list of scanning tools for the session is returned.
- If the context is analyze, a list of analysis tools for the session is returned.
- If the context is to apply fuzzy fusion, a list of fuzzification tools for the session is returned.

3. Choose the session

The list of sessions returned from the database is presented to the IDEA User. The IDEA User has the option to select, from the list of sessions, one session to launch.

5.2.3.3 End of use case

The session information is checked out for this user - the session is locked. This ensures that the tool's data is not modified by other users.

Use case ends.

5.2.3.4 Alternative Flows

5.2.3.4.1 1a. User cancels

In Steps 1, 3, and 5 the user has the option to cancel the operation. If the user chooses to cancel, the use case ends.

5.2.3.4.2 2a. No sessions in the database

In Step 2, if no sessions matching the tool criteria are found within the database the user is informed. This ends this use case.

5.2.3.4.3 2b. Database not found

In Step 2, if there is an error communicating with the database, the event is logged, the user informed and the use case ends.

5.2.4 Post-Conditions

5.2.5 Scenario Diagrams

5.2.5.1 Scenario 1 - Basic flow

Table 5-4 - Open Session Scenario 1

Initiator	Behavior	Target
IDEA User	Select a session to open. Specify read-only or read-write.	IDEA Component
IDEA Component	Query the database for the session.	Repository Component
Repository Component	If the session is locked then we can only open read-only. If the session is unlocked, lock it. Get the session information.	Database

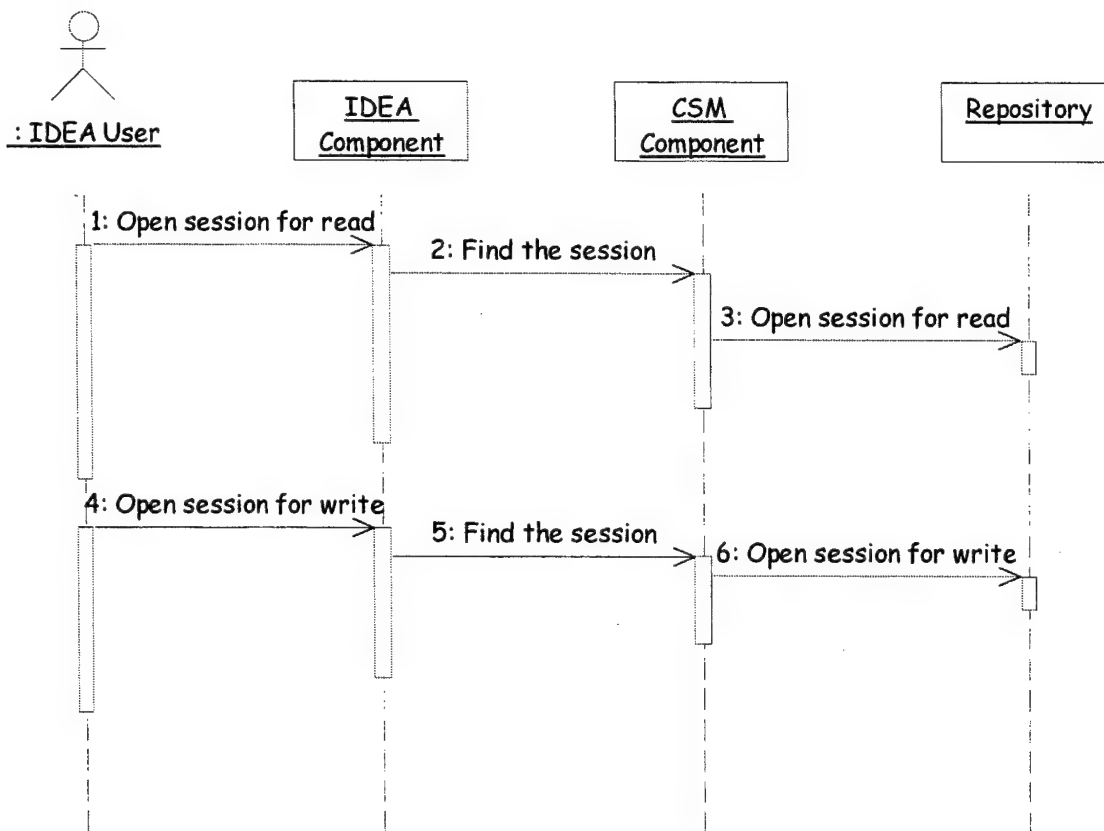


Figure 5-7 - Basic flow

5.2.5.2 Scenario 2 - User cancels

Table 5-5 - Open Session Scenario 2

Initiator	Behavior	Target
IDEA User	User cancels operation.	IDEA Component

Figure - User cancels

5.2.5.3 Scenario 3 - No session in the database

Table 5-6 - Open Session Scenario 3

Initiator	Behavior	Target
IDEA User	Select a session to open. Specify read-only or read-write.	IDEA Component
IDEA Component	Query the database for the session.	Repository Component
Repository Component	Session not found	IDEA Component
IDEA Component	Session not found	IDEA User

Figure - No session in the database

5.2.5.4 Scenario 4 - Database not found

Table 5-7 - Open Session Scenario 4

Initiator	Behavior	Target
IDEA User	Select a session to open. Specify read-only or read-write.	IDEA Component
IDEA Component	Query the database for the session.	Repository Component
Repository Component	Database error	IDEA Component
IDEA Component	Database error	IDEA User

Figure - Database not found

5.3 Use Case Close Session

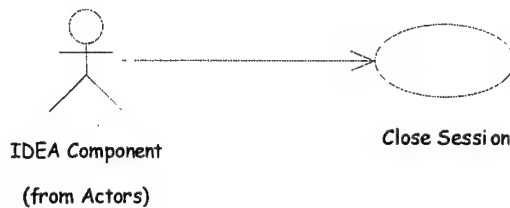


Figure 5-8 - Close Session Use Case

5.3.1 Brief Description

In this use case, the IDEA system closes an opened session.

5.3.2 Pre-Conditions

Before this use case starts, the session must have been opened as detailed in the Open Session use case.

5.3.3 Flow of Events

5.3.3.1 Start of use case

This use case starts when it is necessary to close an opened session. This use case follows session discovery, scanning, analysis, and fuzzy fusion analysis and occurs after the external tools have updated the session's information.

5.3.3.2 Basic Flow

1. Unlock the session

Unlock the session prior to closing it or during the closing of it.

5.3.3.3 End of use case

After the tool information has been stored in the database the user is informed of a successfully completed operation. After the user acknowledges the operation complete the use case ends.

5.3.3.4 Alternative Flows

5.3.3.4.1 Session is not open

If the session is not open, ignore the request.

5.3.4 Post-Conditions

5.3.5 Scenario Diagrams

5.3.5.1 Scenario 1 - Basic flow

Table 5-8 - Close Session Scenario 1

Initiator	Behavior	Target
IDEA Component	Notify the database to unlock the session	Repository Component
Repository Component	Unlock the session	Repository
Repository Component	Session unlocked	IDEA Component
IDEA Component	Session unlocked	IDEA User

Figure - Basic flow

5.3.5.2 Scenario 2 - Session is not open

Table 5-9 - Close Session Scenario 2

Initiator	Behavior	Target
IDEA Component	Notify the database to unlock the session	Repository Component
Repository Component	Unlock the session	Repository
Repository Component	Session not locked	IDEA Component
IDEA Component	Session unlocked	IDEA User

Figure - Session is not open

5.4 Use Case Manage Session

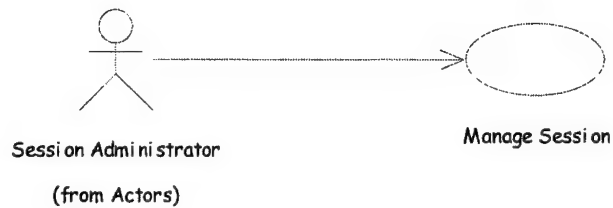


Figure 5-9 - Manage Session Use Case

5.4.1 Brief Description

This use case describes how a user performs CRUD operations on a session.

5.4.2 Pre-Conditions

Configuration of the repository must be completed prior to this use case.

5.4.3 Flow of Events

5.4.3.1 Start of use case

This use case starts when a user elects to create, read, update, or delete a session.

5.4.3.2 Basic Flow

1. Choose Create operation

The user opts to perform one of the following functions. For each function, the user provides a session name. A valid session name may be retrieved using the Get Session List use case, or, for the create session operation, the user provides one.

Create - to create a session, the user enters a session name.

2. Instantiate a session

A message is sent to the Repository component to create a new entry for the session. The name of the session is provided to the Repository.

The Repository, guided by its configuration, creates a new entry for the session.

5.4.3.3 End of use case

This use case ends by performing the Close Session use case.

5.4.3.4 Alternative Flows

1a. Choose Read operation

In step 1, the user selects read instead of create. The session is opened for read and presented to the user.

This use case ends with Close Session.

1b. Choose Update operation

In step 1, the user selects update instead of create. The session, if unlocked, is opened and locked and the user allowed to make changes. After the changes are made the user has the ability to save or discard the changes.

This use case ends with Close Session.

1c. Choose Delete operation

In step 1, the user selects delete instead of create. The session, if unlocked and if its repository state allows deletion, is deleted from the database. The TBI section describes the criteria for deleting sessions.

This use case ends with Close Session.

1d. Session already exists

In step 1, if the user provides a session name that already exists in the database, the user is informed that the session name provided already exists.

1e. Session does not exist

In step 1, or 1a, 1b, or 1c, if the user provides a session name that does not exist in the database, the user is informed and the use case restarts.

1f. Cancel

For Create, the user can cancel before the session is created - prior to step 2.

For Read, the user can cancel prior to the read operation - prior to alternate step 1a.

For Update, the user can cancel prior to saving the updated session - in alternate step 1b.

For Delete, the user can cancel prior to alternate step 1c.

2a. Cannot instantiate

In step 2, if Repository component errors prevent the instantiation of the session the user notified and the use case ends.

5.4.4 Post-Conditions

For Create, the session is created in the database.

For Read, the session is unchanged in the database.

For Update, the session information is updated in the database.

For Delete, the session no longer exists in the database.

5.4.5 Scenario Diagrams

5.4.5.1 Scenario 1 - Basic flow

Table 5-10 - Manage Session Scenario 1

Initiator	Behavior	Target
IDEA Administrator	Open Session Manager	IDEA Component
IDEA Administrator	Perform Create operation on a	IDEA Component

	session	
IDEA Component	Execute the Create operation	IDEA Component
IDEA Administrator	Exit the Session Manager	IDEA Component

Figure - Basic flow

5.4.5.2 Scenario 2 - Choose Read operation

Table 5-11 - Manage Session Scenario 2

Initiator	Behavior	Target
IDEA Administrator	Open Session Manager	IDEA Component
IDEA Administrator	Perform Read operation on a session	IDEA Component
IDEA Component	Execute the Read operation	IDEA Component
IDEA Administrator	Exit the Session Manager	IDEA Component

Figure - Choose Read operation

5.4.5.3 Scenario 3 - Choose Update operation

Table 5-12 - Manage Session Scenario 3

Initiator	Behavior	Target
IDEA Administrator	Open Session Manager	IDEA Component
IDEA Administrator	Perform Update operation on a session	IDEA Component
IDEA Component	Execute the Update operation	IDEA Component
IDEA Administrator	Exit the Session Manager	IDEA Component

Figure - Choose Update operation

5.4.5.4 Scenario 4 - Choose Delete operation

Table 5-13 - Manage Session Scenario 4

Initiator	Behavior	Target
-----------	----------	--------

IDEA Administrator	Open Session Manager	IDEA Component
IDEA Administrator	Perform Delete operation on a session	IDEA Component
IDEA Component	Execute the Delete operation	IDEA Component
IDEA Administrator	Exit the Session Manager	IDEA Component

Figure - Choose Delete operation

5.4.5.5 Scenario 5 - Session already exists

Table 5-14 - Manage Session Scenario 5

Initiator	Behavior	Target
IDEA Administrator	Open Session Manager	IDEA Component
IDEA Administrator	Perform Create operation on a session	IDEA Component
IDEA Component	Execute the Create operation	IDEA Component
Repository Component	Execute the Create operation	Database
Database	Session exists	Repository Component
Repository Component	Session exists	IDEA Component
IDEA Component	Session exists	IDEA Administrator

Figure - Session already exists

5.4.5.6 Scenario 6 - Session does not exist

Table 5-15 - Manage Session Scenario 6

Initiator	Behavior	Target
IDEA Administrator	Open Session Manager	IDEA Component
IDEA Administrator	Perform Read, Update, or Delete operation on a session	IDEA Component
IDEA Component	Execute the Read, Update, or Delete operation	IDEA Component
Database	Session does not exist	Repository Component
Repository Component	Session does not exist	IDEA Component
IDEA Component	Session does not exist	IDEA Administrator

Figure - Session does not exist

5.4.5.7 Scenario 7 - Cancel

Table 5-16 - Manage Session Scenario 7

Initiator	Behavior	Target
IDEA Administrator	Cancel operation	IDEA Component
IDEA Component	End operation	IDEA Component

Figure - Cancel

6 Model Preparation Use Cases

These use cases focus on the initial preparations to make prior to running the System Analysis Use Cases.

Tool running - user define list for tool order

How much leeway will the user have in picking the order?

- 1) Order tools within their category only (discovery, scanner, ...)
- 2) Order tools outside of their category (may have analysis before discovery, etc.)

Tool information (for each tool)

How to map the tool needs to the information contained within the session

How to map the tool results back into the session

6.1 Use Case Configure Tool

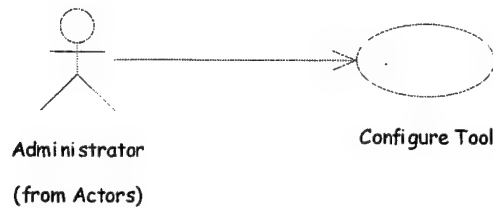


Figure 6-1 - Configure Tool Use Case

6.1.1 Brief Description

Tool configuration is the one-time process of integrating IA tools with IDEA. IDEA stores and maintains the information used by and generated by the IA tools.

This use case provides functionality for new tools to be integrated into the suite of IDEA tools. IDEA requires launch behavior for the tool. IDEA also maps the session information to the tools data format. Each tool requires a different launch protocol, a different data format, and different CSM information. Tool configuration consists of identifying CSM information required by a tool, the format of the data, and the method for launching the tool with the information.

During tool configuration the IDEA Administrator decides what additional information the tool will require and tailors how it will collect the information (from another tool, from the user).

6.1.2 Pre-Conditions

There are four tool types - Discovery, Scanner, Analysis, Fuzzy Fusion

The CSM information exists.

If required, a mapping between CSM model and Tool model exists.

6.1.3 Flow of Events

6.1.3.1 Start of use case

The IDEA Administrator chooses to open the tool configuration manager.

The tool configuration manager allows the administrator to configure new tools for IDEA. The tool information is filled out and saved to the repository.

6.1.3.2 Basic flow

1. Select the type of tool to configure

The IDEA Administrator chooses the type of tool to configure: one of Discovery, Scan, Analysis, or Fuzzy Analysis.

2. Get the tool information

The tool information dialog allows the IDEA Administrator to insert information about the tool. Refer to *the appendix on tool information* for the fields.

3. Select the CSM information needed by the tool

The IDEA Administrator selects which repository information is needed by the tool and selects it.

4. Select the Tool to Repository component interaction protocol

The IDEA Administrator chooses the protocol by which the tool will interact with the repository.

5. Save the tool configuration

The IDEA Administrator saves the tool configuration.

6.1.3.3 End of use case

This use case ends when the tool configuration is saved to the repository.

6.1.3.4 Alternative flows

1a. User cancels

In step 1, 2, 3, 4, and 5 the IDEA Administrator has the option to cancel the operation. The use case ends.

2a. No tool information exists

In step 2, if the tool information for the type of tool selected by the IDEA Administrator is unavailable within the repository, the IDEA Administrator is notified and the use case ends.

2b. No database available

In step 2, if the repository is unavailable, the IDEA Administrator is notified of the problem and the use case ends.

3a. No database available

In step 3, if the repository is unavailable, the IDEA Administrator is notified of the problem and the use case ends.

6.1.4 Special Requirements

6.1.5 Post-Conditions

6.1.6 Scenarios

6.2 Use Case Assign Tool

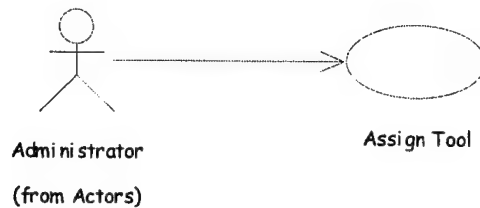


Figure 6-2 - Assign Tool Use Case

6.2.1 Brief Description

The IDEA Administrator has the capability to assign configured tools to a session. After a session is identified, a tool is selected and added to the session's tool list. Multiple tools may be selected. The session is saved after tool selection is complete.

6.2.2 Pre-Conditions

Sessions exist within the database.

Tools that will be assigned have been configured.

6.2.3 Flow of Events

6.2.3.1 Start of use case

This use case starts when the IDEA Administrator selects the assign tool operation from the Repository API.

6.2.3.2 Basic flow

1. Open the Repository session list

Get a list of sessions from the repository.

2. Select session to configure

Select, from the list, a session to configure.

3. Open tool List

Get a list of configured tools from the repository.

4. Select tool(s) for the session

Select the tools to apply to the session. Add them to the session list.

5. Specify tool execution order

Specify when each tool will be executed.

6. Close session configuration

Save the session configuration to the repository.

6.2.3.3 End of use case

This use case ends when the session has been saved.

6.2.3.4 Alternative flows

1a. User cancels

In step 1, 2, 3, 4, 5, and 6 the IDEA Administrator has the option to cancel the operation. The use case ends.

2a. No tool information exists

In step 2, if the tool information for the type of tool selected by the IDEA Administrator is unavailable within the repository, the IDEA Administrator is notified and the use case ends.

2b. No database available

In step 2, if the repository is unavailable, the IDEA Administrator is notified of the problem and the use case ends.

3a. No database available

In step 3, if the repository is unavailable, the IDEA Administrator is notified of the problem and the use case ends.

6.2.4 Special Requirements

6.2.5 Post-Conditions

6.2.6 Scenarios

6.3 Use Case Assign Profile

Figure 6-3 - Assign Profile Use Case

6.3.1 Brief Description

The IDEA Administrator has the capability to assign security profiles to a session. After a session is identified, a profile is selected and added to the session's profile list. Multiple tools may be selected. The session is saved after tool selection is complete.

6.3.2 Pre-Conditions

Sessions exist within the repository.

Profiles that will be assigned have been defined.

6.3.3 Flow of Events

6.3.3.1 Start of use case

This use case starts when the IDEA Administrator selects the assign profile operation from the Repository API.

6.3.3.2 Basic flow

1. Open the Repository session list

Get a list of sessions from the repository.

2. Select session to configure

Select, from the list, a session to configure.

3. Open profile List

Get a list of pre-defined profiles from the repository.

4. Select profile(s) for the session

Select the profile(s) to apply to the session. Add them to the session list.

5. Close session configuration

Save the session configuration to the repository.

6.3.3.3 End of use case

This use case ends when the session has been saved.

6.3.3.4 Alternative flows

1a. User cancels

In step 1, 2, 3, 4, 5, and 6 the IDEA Administrator has the option to cancel the operation. The use case ends.

2a. No tool information exists

In step 2, if the tool information for the type of tool selected by the IDEA Administrator is unavailable within the repository, the IDEA Administrator is notified and the use case ends.

2b. No database available

In step 2, if the repository is unavailable, the IDEA Administrator is notified of the problem and the use case ends.

3a. No database available

In step 3, if the repository is unavailable, the IDEA Administrator is notified of the problem and the use case ends.

6.3.4 Special Requirements

6.3.5 Post-Conditions

6.3.6 Scenarios

6.4 Use Case Prompt Analyst

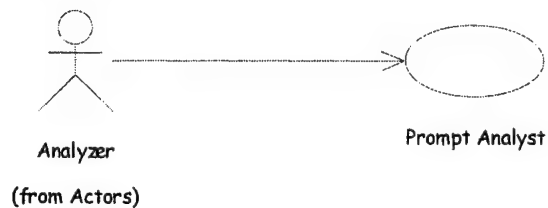


Figure 6-4 - Prompt Analyst Use Case

6.4.1 Brief Description

Prompt the analyst for addition information needed to fill out the CSM or other tool-specific information prior to running scan or analysis tools.

6.4.2 Flow of Events

6.4.2.1 Basic Flow

Table of Actor Steps and Use Case Support

Actor Steps	Use Case Support
	1. Analyze missing data 2. Open dialog 3. Ingest dialog results

6.4.2.2 Alternative Flows

6.4.3 Special Requirements

6.4.4 Pre-Conditions

6.4.5 Post-Conditions

6.4.6 Scenarios

7 Repository Sessions

Within the Repository Component is the database. This database is used to store session information (system models) defined by a user and filled with a discovery tool. The session information is provided to scan and analysis tools. The results of the tools go back into the session as tool results. Fuzzy fusion takes these tool results and generates a system vulnerability level.

A session is a named (internally-named) set of data records that comprise a model, a suite of tools with which to test the model, and the results of those tests.

There are three stages to the session.

<Name>.a.b

Where <Name> represents the name of the session. ".a" is represented as a integer value and its presence implies that a model is defined either by running a discovery tool or manual entry. ".b" is represented as a integer value and its presence implies that a suite of tools has been assigned to analyze the model. There may be tool results available for a <Name>.a.b session.

7.1 Schemas

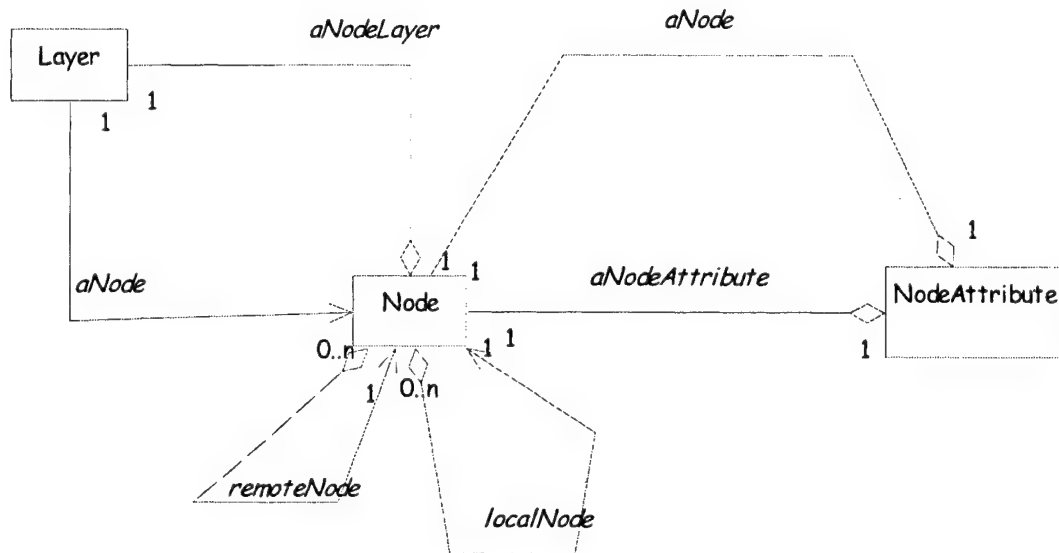


Figure 7-1 - Data Model Diagram

7.1.1 Layer

Table 7-1 - Layer

Entry	Description
Name	String representing the layer's unique name.

Layer Type	The type of the layer
Node Type List	A list of the types of nodes which a layer may contain.

The type of layer determines possible node types that can exist on it. This table is a mapping for layer types to node types.

Layers defined of a particular type can contain only nodes of the type in the Node Type List.

7.1.2 Node

Table 7-2 - Node

Entry	Description
Name	String representing the node's unique name
Node Type	The type of the node
Remote Node Type List	List of nodes this node interacts with that are not part of this layer.
Attribute Type	The type of attribute a Node can have

The type of node determines the node network syntax. Nodes of a certain type can only be linked with nodes of a certain type. This table is a mapping of which nodes can be linked to other nodes.

7.1.3 Attributes

Table 7-3 - Attributes

Entry	Description
Name	String representing a unique name.
Type	
State	

8 Logical View

This section describes the static view of IDEA showing the component-level use cases for the IDEA, Repository, and Tool components and the class diagrams supporting the use cases.

8.1 Threads - Component-Level Use Cases

This section describes the component-level use cases, termed software threads. Software threads describe the behavior of each component as it responds to external commands via the component's interface.

Each of the three components, IDEA, Repository, and Tool, define an interface that clients of the component may use to invoke component behavior. IDEA Users interact with the IDEA application using the IDEA Component's API. The IDEA Component API provides operations to support network design and analysis including Fuzzy Fusion analysis.

The API of the Tool component is designed solely for use by the IDEA component. The Repository component API is required by both the IDEA and Tool components as well as a possible Repository GUI.

The threads described below detail the class and data interactions of each API.

8.2 Use Case to Thread Mapping

The following table maps the software threads described in this section to the system-level use cases described above.

Table 8-1

Use Case	Thread	QC
Launch Tool	Get Investigation List Open Investigation Start Investigation Save Investigation Close Investigation	
Launch Discover Tool	Get Investigation List Open Investigation Start Investigation Save Investigation Close Investigation	
Launch Scan Tool	Get Investigation List Open Investigation Start Investigation Save Investigation Close Investigation	
Launch Analysis Tool	Get Investigation List Open Investigation	

	Start Investigation Save Investigation Close Investigation	
Launch Fuzzy Fusion Analysis	Get Investigation List Open Investigation Start Investigation Save Investigation Close Investigation	
View Results	Get Investigation List View Investigation	
Get Session List	Fetch Session List	
Open Session	Open Session	
Close Session	Close Session	
Manage Session	Save Session View Session Copy Session Delete Session Rename Session	
Configure Tool	TBI	
Assign Tool	Fetch Session List Open Session Fetch Tool Information Assign Tool Save Session Close Session	
Assign Security Profile	Assign Profile	
Prompt Analyst	TBI	

8.3 IDEA API Threads

8.3.1 Get Investigation List

8.3.1.1 Brief Description

This use case allows clients external to the IDEA API (IDEA User) to get a list of possible investigations set up in the Common System Model.

8.3.1.2 Pre-Conditions

System configured.

8.3.1.3 Flow of Events

8.3.1.3.1 Start of use case

This use case starts when the client requests a list of investigations from IDEA. The client passes as a parameter the kind of investigation required. Investigation kinds are: *Map*, *Populate*, *Analysis*, and *Fuse*. The kind *All* is used to request all investigations.

8.3.1.3.2 Basic flow

1. Get the session list

Get the session information from the Repository Component.

8.3.1.3.3 End of use case

This use case ends when the client receives the list of investigations.

8.3.1.3.4 Alternative flows

1a. Investigation not found

In step 1, if no list of investigations is found, status is returned. The use case ends.

1b. Database error

If there is a problem retrieving the list of investigations, status is returned. The use case ends.

8.3.1.4 Post-Conditions

The client has a list of strings representing the names of the investigations of the requested kind that are available within the repository.

8.3.1.5 Special Requirements

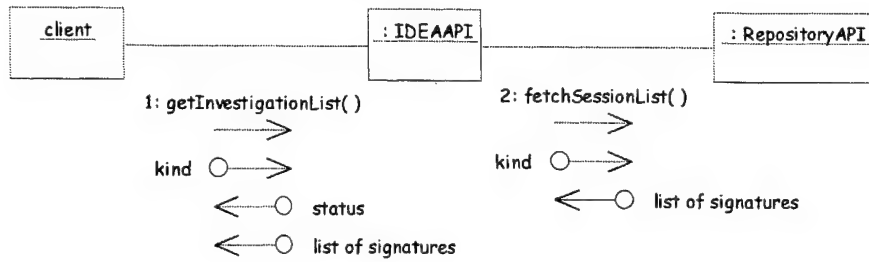
1. Return status

Table 8-2

Returned Value	Function
	Database Error

	Investigation Not Found
	Successful Completion

8.3.1.6 Collaboration Diagrams



8.3.2 Open Investigation

8.3.2.1 Brief Description

This use case is responsible for allowing clients to open an existing investigation for mapping, populating, analyzing, or fusing.

8.3.2.2 Pre-Conditions

System configured.

The investigation exists.

8.3.2.3 Flow of Events

8.3.2.3.1 Start of use case

This use case starts when the clients calls the `openInvestigation` method of the `IDEAAPI`. The client passes, via parameters, the string name of the investigation.

8.3.2.3.2 Basic flow

1. Create an investigation instance

An object to store the investigation information is instantiated.

2. Get session

The session information is retrieved from the Repository Component. The Repository Component returns the session information wrapped within an instance of the `Session` class.

3. Initialize the investigation instance

The investigation object is initialized with the session information.

4. Set the focus of the investigation instance

The focus of the investigation instance relates to the type of tools that will be launched when the *launch* method is used. The focus is one of: *Map*, *Populate*, *Analyze*, *Fuse*, or *All*.

8.3.2.3.3 End of use case

This use case ends when status is returned notifying the client of the success or failure of the operation.

8.3.2.3.4 Alternative flows

2a. Session not found in the repository

In step 2, the session that was requested was not found in the repository. Status is returned and the use case ends.

2b. Database error

If there is a problem retrieving the list of investigations, status is returned. The use case ends.

8.3.2.4 Post-Conditions

The Investigation instance is ready to launch tools.

8.3.2.5 Special Requirements

1. Control parameters

There are five parameters for controlling the behavior of an investigation.

Table 8-3

Control Parameter	Function
Map	Setup the network by adding the network information to the investigator.
Populate	Scan the network for configuration information and fill out the investigation.
Analyze	Analyze the network information for vulnerabilities.
Fuse	Run Fuzzy Fusion analysis on the analysis information and update the investigation.
All	Run the full suite of capabilities for the investigation.

2. Return status

Table 8-4

Returned Value	Function
	Database Error
	Information Not Found
	Successful Completion

8.3.2.6 Collaboration Diagrams

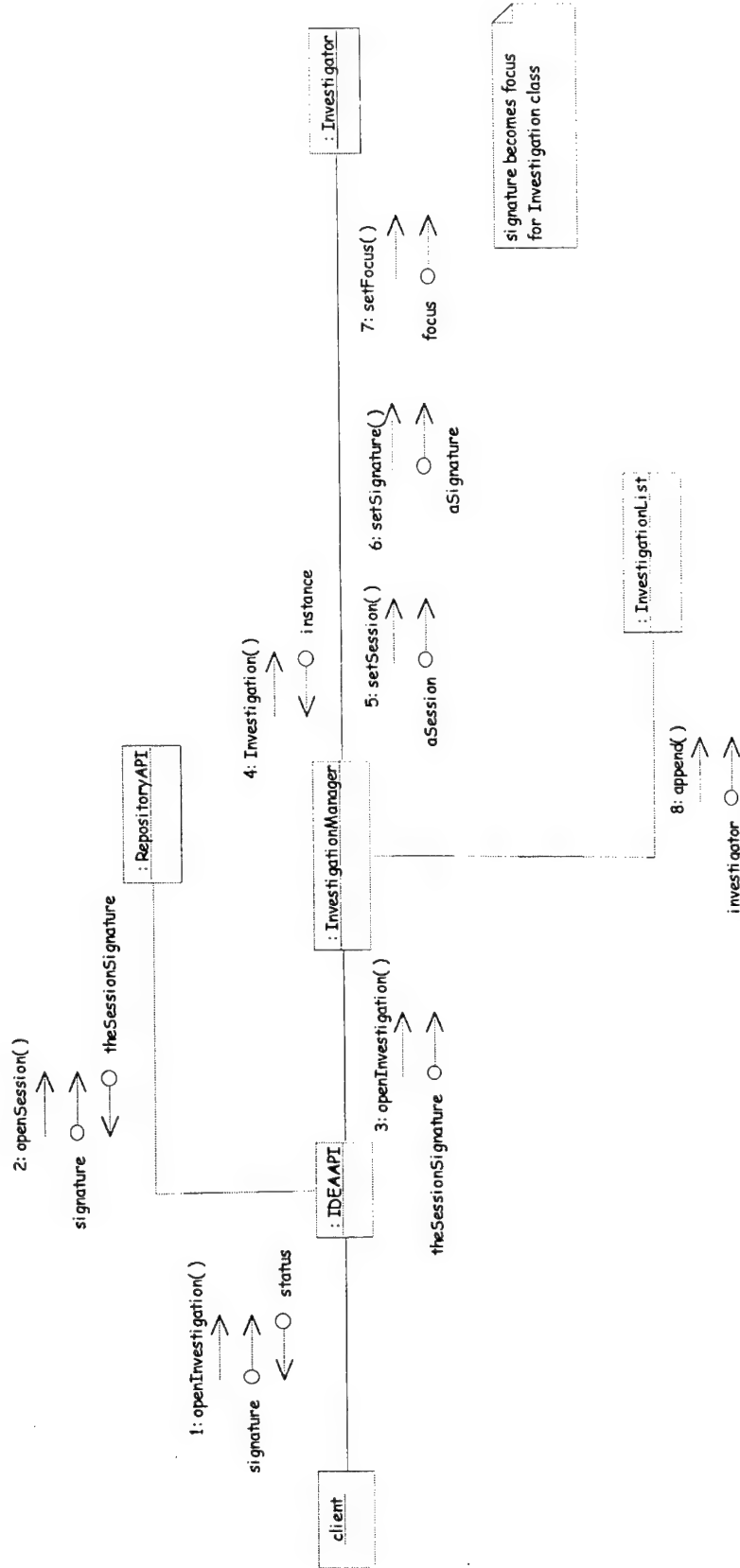


Figure 8-1 - Open Investigation

8.3.3 Start Investigation

8.3.3.1 Brief Description

This use case allows the client to launch COTS or other external tools for the purposes of mapping, populating, analyzing, or fusing the session information.

8.3.3.2 Pre-Conditions

The investigation has been opened.

8.3.3.3 Flow of Events

8.3.3.3.1 Start of use case

This use case starts when the client calls the IDEAAPI startInvestigation method.

8.3.3.3.2 Basic flow

1. Launch the tool

The IDEAAPI has the investigation manager launch the investigation. The investigation tells its session to launch. The session gets the list of possible tools to launch for its particular focus (Map, Populate, Analyze, Fuse, or All) and, one tool at a time, launches it.

8.3.3.3.3 End of use case

This use case ends when the client is informed of the status of the startInvestigation operation.

8.3.3.3.4 Alternative flows

1a. No tool to launch

In step 1, if there are no tools to launch, status is returned and the use case ends.

1b. Unable to launch tool

In step 1, if there is insufficient information to launch the tool, or for any other reason (i.e. incorrect configuration) status is returned and the use case ends.

8.3.3.4 Post-Conditions

8.3.3.5 Special Requirements

1. Return status

Table 8-5

Returned Value	Function
----------------	----------

	No Launch Tool
	Unable to Launch Tool
	Successful Completion

8.3.3.6 Collaboration Diagrams

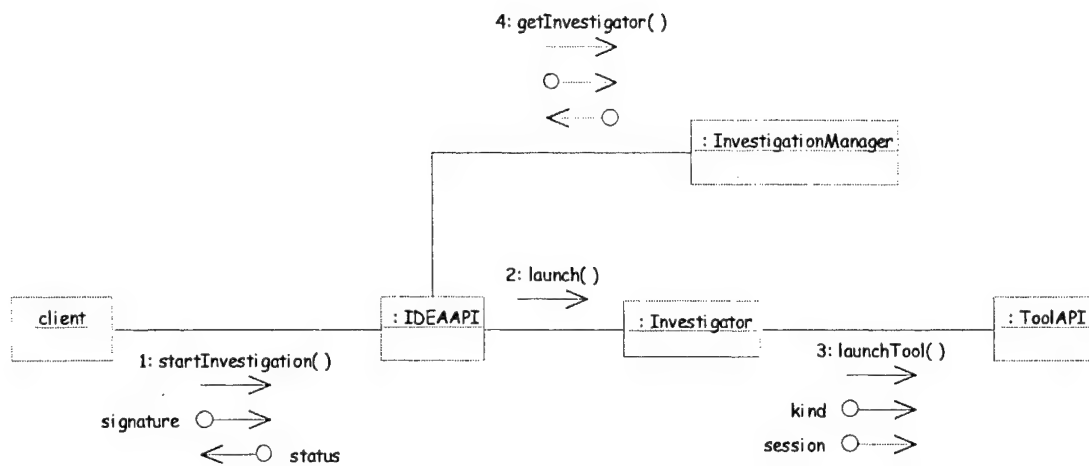


Figure 8-2 - Start Investigation

8.3.4 Save Investigation

8.3.4.1 Brief Description

This use case allows the client to save new information gathered after an investigation has been started. The new information will be stored into the database.

8.3.4.2 Pre-Conditions

The investigation has been started.

8.3.4.3 Flow of Events

8.3.4.3.1 Start of use case

This use case starts when the client calls the IDEAAPI saveInvestigation method.

8.3.4.3.2 Basic flow

1. Save investigation

The IDEAAPI has the investigation manager save the investigation. The investigation tells its session to save the information. The session saves its data to the repository.

8.3.4.3.3 End of use case

This use case ends when the client is informed of the status of the saveInvestigation operation.

8.3.4.3.4 Alternative flows

1a. Unable to save

If there is a problem saving the information, the client is informed via the return status of the operation.

8.3.4.4 Post-Conditions

Successful execution of this use case will ensure that the investigation's information is saved with the repository.

8.3.4.5 Special Requirements

1. Return status

Table 8-6

Returned Value	Function
	Unable to Save
	Successful Completion

8.3.4.6 Collaboration Diagrams

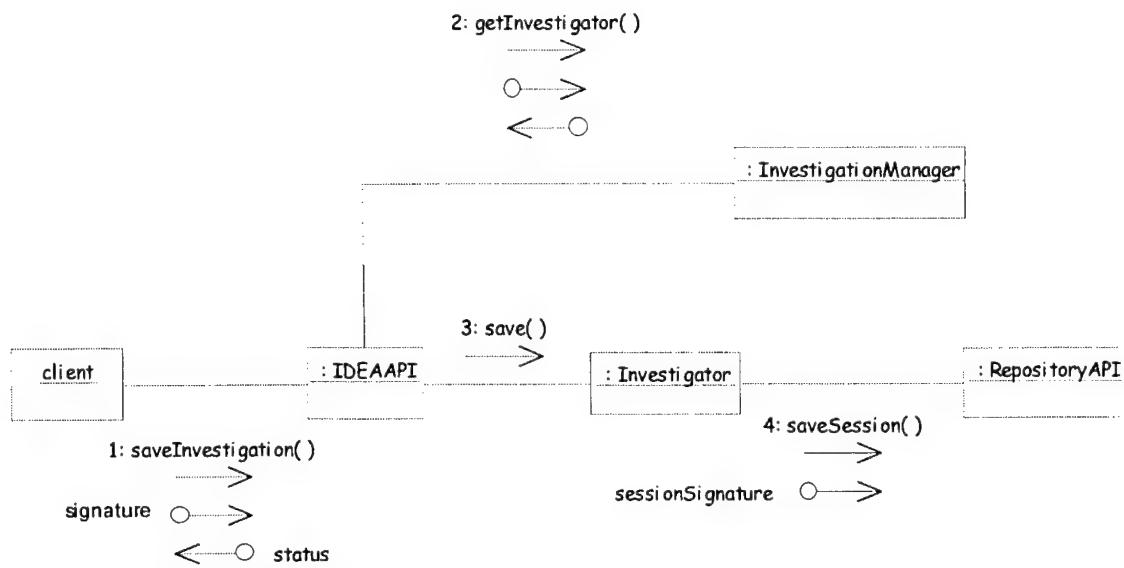


Figure 8-3 - Start Investigation

8.3.5 Close Investigation

8.3.5.1 Brief Description

This use case allows the client to close the investigation.

8.3.5.2 Pre-Conditions

The investigation exists within the database.

8.3.5.3 Flow of Events

8.3.5.3.1 Start of use case

This use case starts when the client calls the IDEAAPI closeInvestigation method.

8.3.5.3.2 Basic flow

1. Close the investigation

The IDEAAPI has the investigation manager close the investigation. The close behavior is propagated to the database where the session information was locked.

8.3.5.3.3 End of use case

This use case ends when status is returned to the client.

8.3.5.3.4 Alternative flows

1a. Investigation not open

If the investigation is not open status is returned and the use case ends.

1b. Not owner.

If the client is not the owner of the investigation, status is returned and the use case ends.

1c. Not saved.

If the investigation has not been saved yet, the client will have status returned showing that the investigation was not unlocked.

8.3.5.4 Post-Conditions

The investigation information is unlocked.

8.3.5.5 Special Requirements

1. Unlock Session

If the investigation was opened for writing the investigations data will be unlocked.

2. Control parameters

There shall be parameters to control the closing of the investigation.

Table 8-7 Control Parameters

Control Parameter	Function
Force Close	<p>If the investigation is not saved, or was left open, the presence of this parameter allows the investigation to be successfully closed.</p> <p>If the investigation's repository is locked, this parameter will allow the database to, if possible, unlock the data.</p>
Don't Save	The presence of this parameter allows the investigation to close the database without updating the data.

3. Return status

Table 8-8

Returned Value	Function
	Investigation Not Open
	Insufficient Privileges
	Not Saved, Confirmation to Close Required
	Successful Completion

8.3.5.6 Collaboration Diagrams

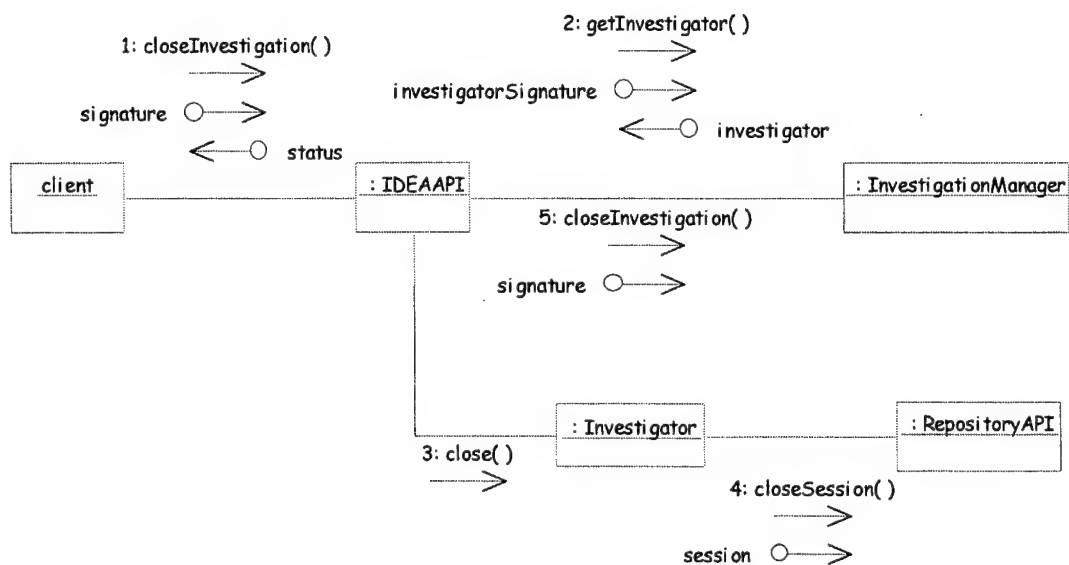


Figure 8-4 - Close Investigation

8.3.6 View Investigation

8.3.6.1 Brief Description

This use case allows the client to view the results of an investigation.

There are two different threads for viewing an investigation, one is if the session is empty and requires information from the database. The other thread occurs after the session has retrieved tool results (after the startInvestigation operation).

8.3.6.2 Pre-Conditions

The client has picked out the investigation to view from an investigation list.

8.3.6.3 Flow of Events

8.3.6.3.1 Start of use case

This use case starts when the client calls the IDEA API viewInvestigation method.

8.3.6.3.2 Basic flow

1. Fetch investigation data

The investigation manager launches an investigation and gets session data for it from the repository.

2. View information

The information is placed in a location designated by the user.

8.3.6.3.3 End of use case

This use case ends when the information is written out for the client and status is returned.

8.3.6.3.4 Alternative flows

1a. Session already exists

In step 1, if the session exists and already has the information within it the investigator sends it the view message and the use case continues to step 2.

1b. No information for the investigation

In step 1, if there is no information available in the database for the investigation, status is returned and the use case ends.

1c. No investigation

In step 1, if the investigation does not exist, status is returned and the use case ends.

1d. Database error

In step 1, if there are problems getting the information from the database, status is returned and the use case ends.

8.3.6.4 Post-Conditions

8.3.6.5 Special Requirements

1. Control parameters

The control parameters of this use case allow the client to specify which

Table 8-9

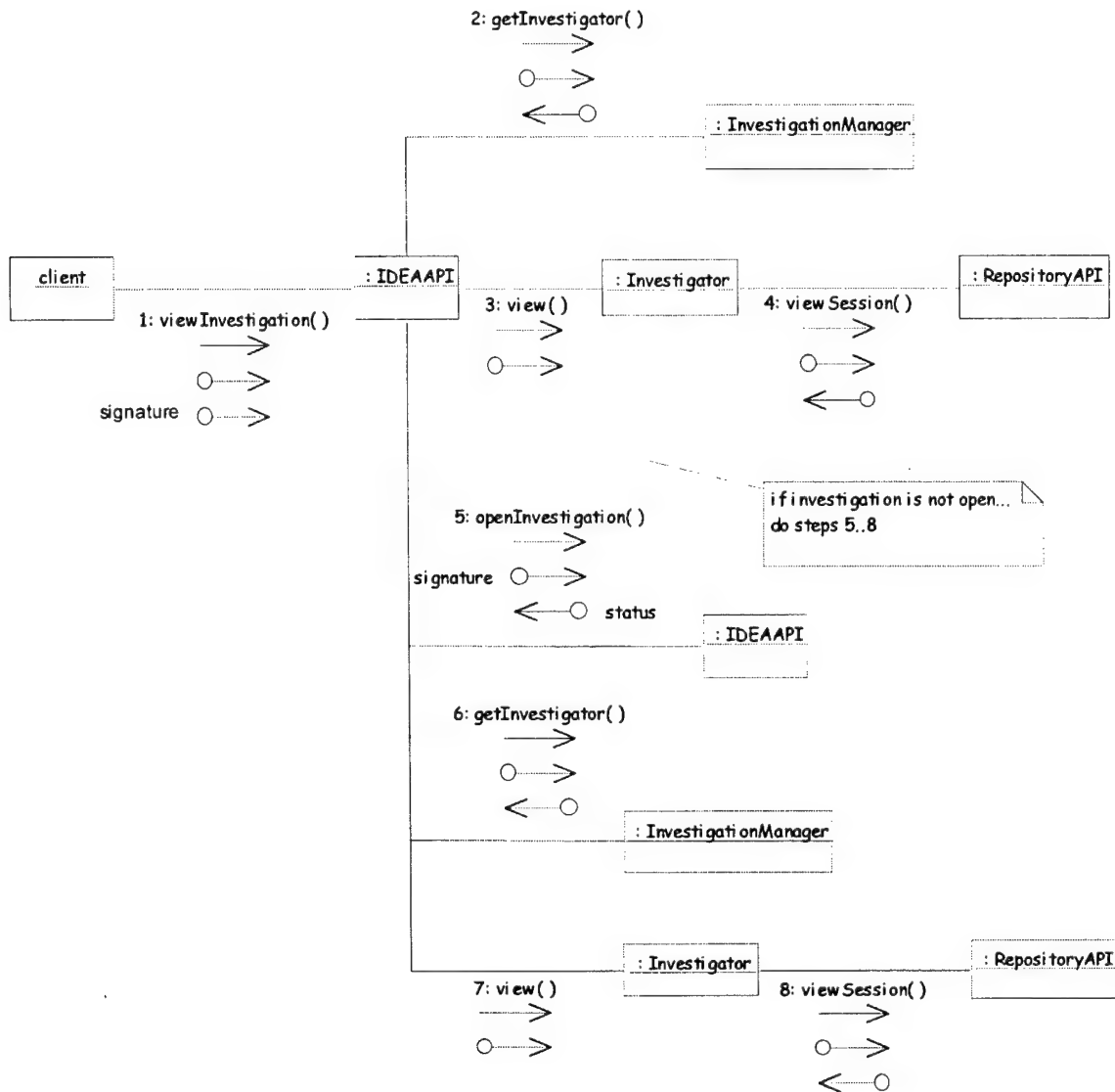
Control Parameter	Function
View Map Results	
View Populate Results	
View Analysis Results	
View Fuse Results	
View All Results	

2. Return status

Table 8-10

Returned Value	Function
	Database Error
	No information for the investigation
	No investigation
	Successful Completion

8.3.6.6 Collaboration Diagrams



8.4 Repository API Threads

8.4.1 Thread Close Session

8.4.1.1 Brief Description

8.4.1.2 Pre-Conditions

8.4.1.3 Flow of Events

8.4.1.3.1 Start of use case

8.4.1.3.2 Basic flow

8.4.1.3.3 End of use case

8.4.1.3.4 Alternative flows

8.4.1.4 Post-Conditions

8.4.1.5 Special Requirements

1. Return status

Table 8-11

Returned Value	Function
	Database Error
	No information for the investigation
	No investigation
	Successful Completion

8.4.1.6 Collaboration Diagrams

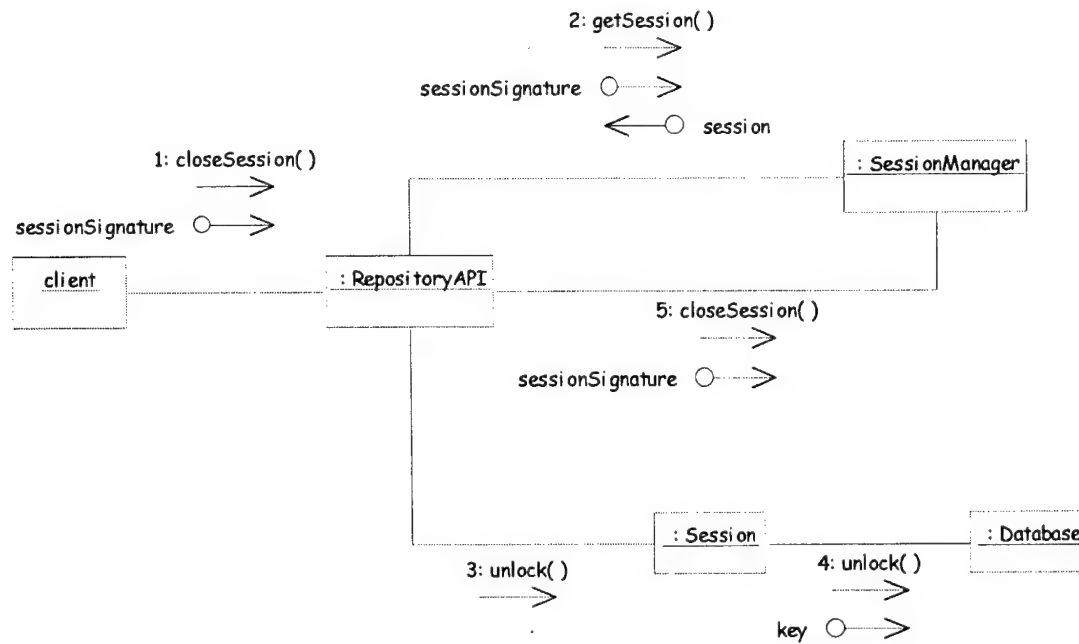


Figure 8-5 - Close Session

8.4.2 Thread Create Session

8.4.2.1 Brief Description

This use case is responsible for allowing clients to create new sessions.

8.4.2.2 Pre-Conditions

8.4.2.3 Flow of Events

8.4.2.3.1 Start of use case

8.4.2.3.2 Basic flow

8.4.2.3.3 End of use case

8.4.2.3.4 Alternative flows

8.4.2.4 Post-Conditions

8.4.2.5 Collaboration Diagrams

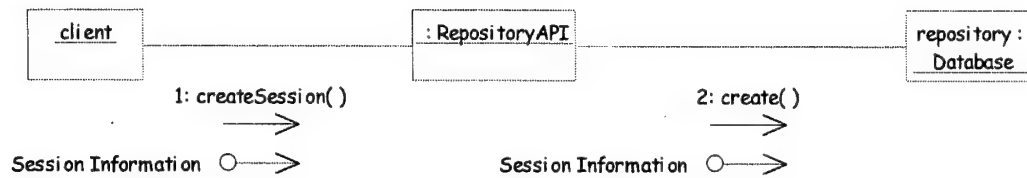


Figure 8-6 - Create Session

8.4.3 Thread Fetch Session List

8.4.3.1 Brief Description

8.4.3.2 Pre-Conditions

8.4.3.3 Flow of Events

8.4.3.3.1 Start of use case

8.4.3.3.2 Basic flow

8.4.3.3.3 End of use case

8.4.3.3.4 Alternative flows

8.4.3.4 Post-Conditions

8.4.3.5 Special Requirements

1. Return status

Table 8-12

Returned Value	Function
	Database Error
	No information for the investigation
	No investigation
	Successful Completion

8.4.3.6 Collaboration Diagrams

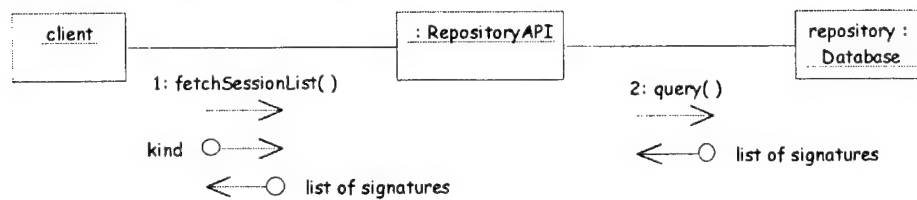


Figure 8-7 - Fetch Session List

8.4.4 Thread Fetch Tool Information

8.4.4.1 Brief Description

8.4.4.2 Pre-Conditions

8.4.4.3 Flow of Events

8.4.4.3.1 Start of use case

8.4.4.3.2 Basic flow

8.4.4.3.3 End of use case

8.4.4.3.4 Alternative flows

8.4.4.4 Post-Conditions

8.4.4.5 Special Requirements

1. Return status

Table 8-13

Returned Value	Function
	Database Error
	No information for the investigation
	No investigation
	Successful Completion

8.4.4.6 Collaboration Diagrams

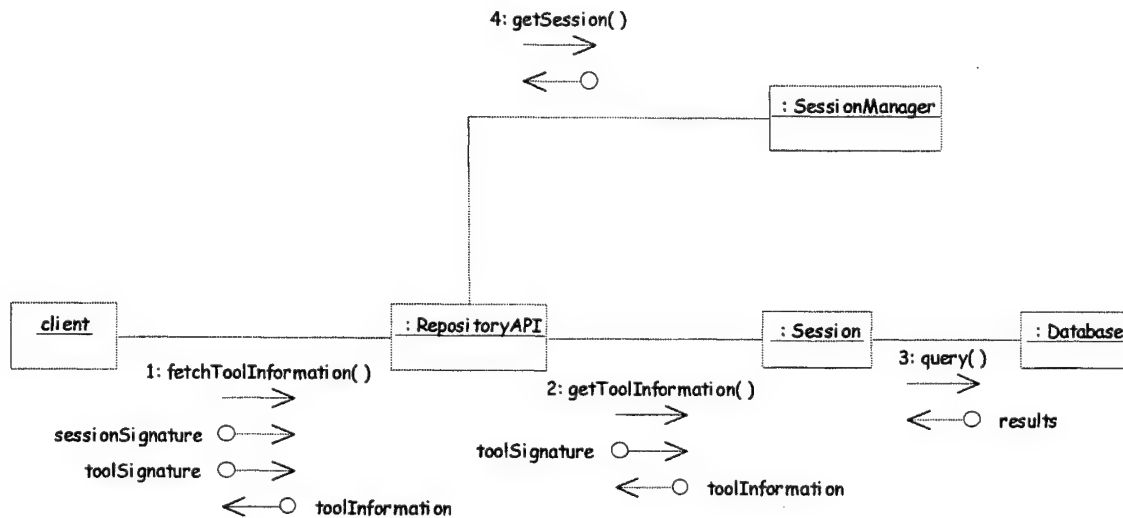


Figure 8-8 - Fetch Tool Information

8.4.5 Thread Fetch Tool List

8.4.5.1 Brief Description

8.4.5.2 Pre-Conditions

8.4.5.3 Flow of Events

8.4.5.3.1 Start of use case

8.4.5.3.2 Basic flow

8.4.5.3.3 End of use case

8.4.5.3.4 Alternative flows

8.4.5.4 Post-Conditions

8.4.5.5 Special Requirements

1. Return status

Table 8-14

Returned Value	Function
	Database Error
	No information for the investigation
	No investigation
	Successful Completion

8.4.5.6 Collaboration Diagrams

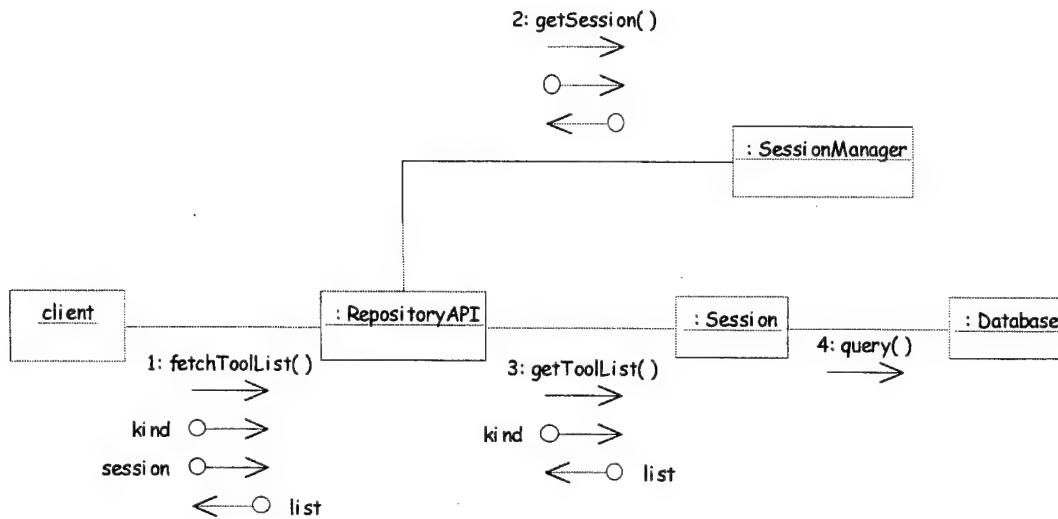


Figure 8-9 - Fetch Tool List

8.4.6 Thread Open Session

8.4.6.1 Brief Description

8.4.6.2 Pre-Conditions

8.4.6.3 Flow of Events

8.4.6.3.1 Start of use case

8.4.6.3.2 Basic flow

8.4.6.3.3 End of use case

8.4.6.3.4 Alternative flows

8.4.6.4 Post-Conditions

8.4.6.5 Special Requirements

1. Return status

Table 8-15

Returned Value	Function
	Database Error
	No information for the investigation
	No investigation
	Successful Completion

8.4.6.6 Collaboration Diagrams

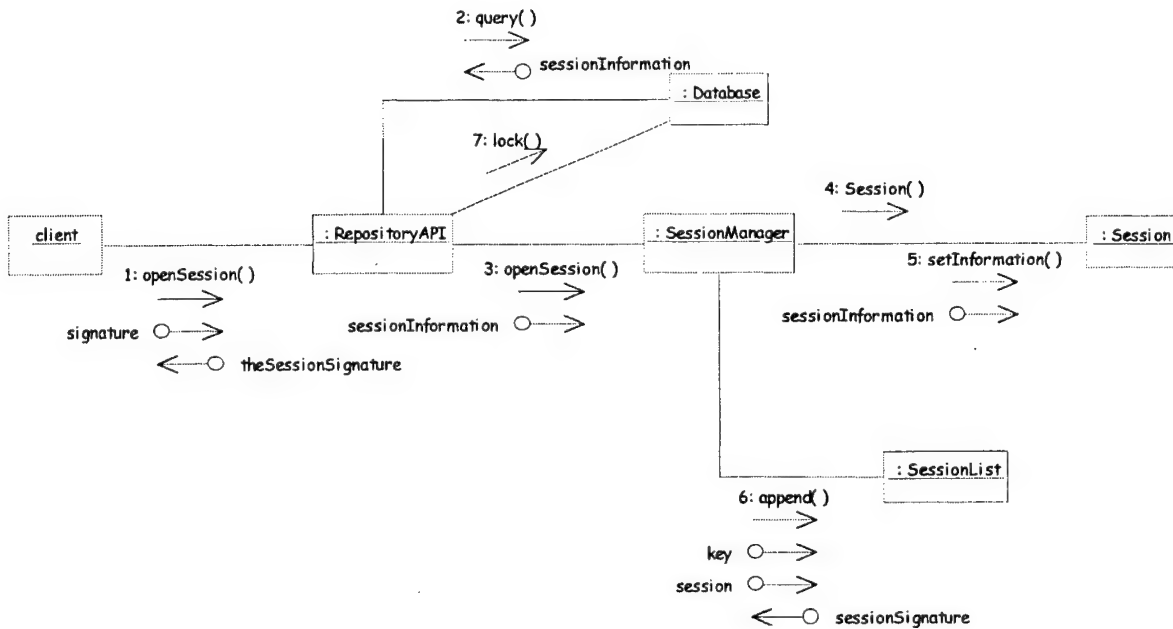


Figure 8-10 - Open Session

8.4.7 Thread Save Session

8.4.7.1 Brief Description

This use case is responsible for allowing clients to save an existing session.

8.4.7.2 Pre-Conditions

8.4.7.3 Flow of Events

8.4.7.3.1 Start of use case

8.4.7.3.2 Basic flow

8.4.7.3.3 End of use case

8.4.7.3.4 Alternative flows

8.4.7.4 Post-Conditions

8.4.7.5 Collaboration Diagrams

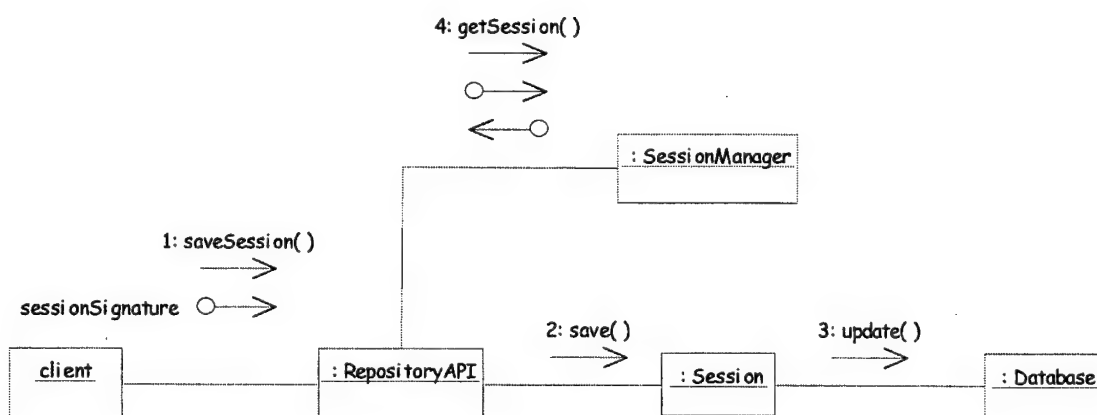


Figure 8-11 - Save Session

8.4.8 Thread View Session

8.4.8.1 Brief Description

This use case is responsible for allowing clients to view an existing session.

8.4.8.2 Pre-Conditions

8.4.8.3 Flow of Events

8.4.8.3.1 Start of use case

8.4.8.3.2 Basic flow

8.4.8.3.3 End of use case

8.4.8.3.4 Alternative flows

8.4.8.4 Post-Conditions

8.4.8.5 Collaboration Diagrams

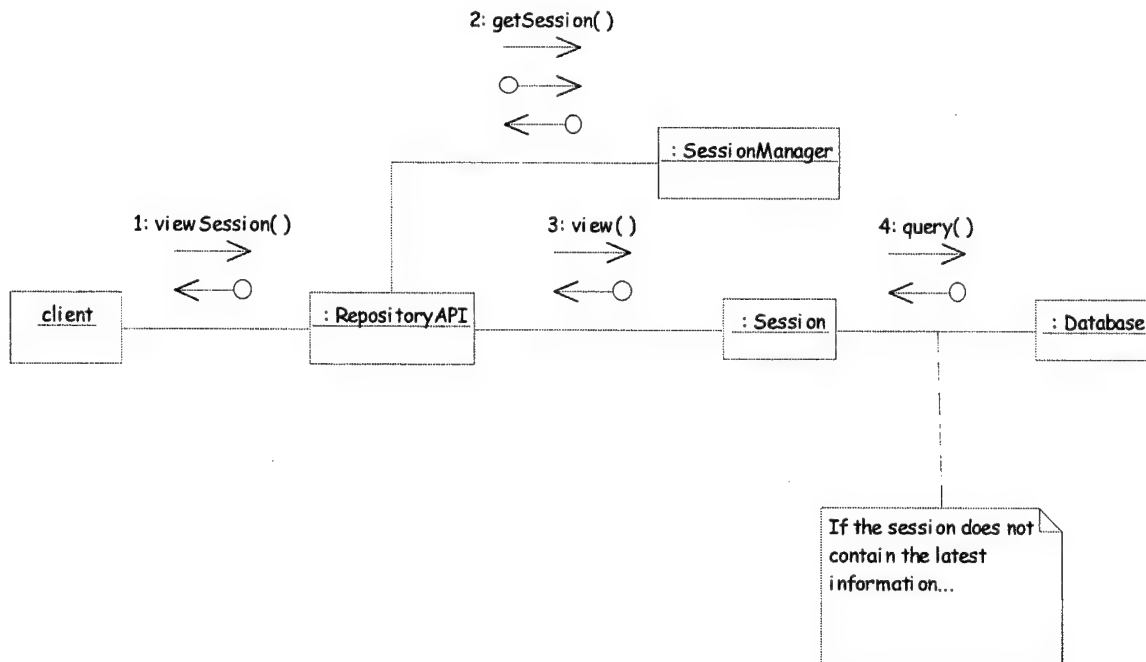


Figure 8-12 - View Session

8.4.9 Thread Copy Session

8.4.9.1 Brief Description

This use case is responsible for allowing clients to copy an existing session to a new session.

8.4.9.2 Pre-Conditions

8.4.9.3 Flow of Events

8.4.9.3.1 Start of use case

8.4.9.3.2 Basic flow

8.4.9.3.3 End of use case

8.4.9.3.4 Alternative flows

8.4.9.4 Post-Conditions

8.4.9.5 Collaboration Diagrams

8.4.10 Thread Delete Session

8.4.10.1 Brief Description

This use case is responsible for allowing clients to delete sessions.

8.4.10.2 Pre-Conditions

8.4.10.3 Flow of Events

8.4.10.3.1 Start of use case

8.4.10.3.2 Basic flow

8.4.10.3.3 End of use case

8.4.10.3.4 Alternative flows

8.4.10.4 Post-Conditions

8.4.10.5 Collaboration Diagrams

8.4.11 Thread Rename Session

8.4.11.1 Brief Description

This use case is responsible for allowing clients to rename existing sessions.

8.4.11.2 Pre-Conditions

8.4.11.3 Flow of Events

8.4.11.3.1 Start of use case

8.4.11.3.2 Basic flow

8.4.11.3.3 End of use case

8.4.11.3.4 Alternative flows

8.4.11.4 Post-Conditions

8.4.11.5 Collaboration Diagrams

8.4.12 Thread Fetch Session Information

8.4.12.1 Brief Description

8.4.12.2 Pre-Conditions

8.4.12.3 Flow of Events

8.4.12.3.1 Start of use case

8.4.12.3.2 Basic flow

8.4.12.3.3 End of use case

8.4.12.3.4 Alternative flows

8.4.12.4 Post-Conditions

8.4.12.5 Special Requirements

1. Return status

Table 8-16

Returned Value	Function
	Database Error
	No information for the investigation
	No investigation
	Successful Completion

8.4.12.6 Collaboration Diagrams

8.5 Tool API Threads

8.5.1 Thread Launch Tool

8.5.1.1 Brief Description

8.5.1.2 Pre-Conditions

8.5.1.3 Flow of Events

8.5.1.3.1 Start of use case

8.5.1.3.2 Basic flow

8.5.1.3.3 End of use case

8.5.1.3.4 Alternative flows

8.5.1.4 Post-Conditions

8.5.1.5 Special Requirements

1. Return status

Table 8-17

Returned Value	Function
	Database Error
	No information for the investigation
	No investigation
	Successful Completion

8.5.1.6 Collaboration Diagrams

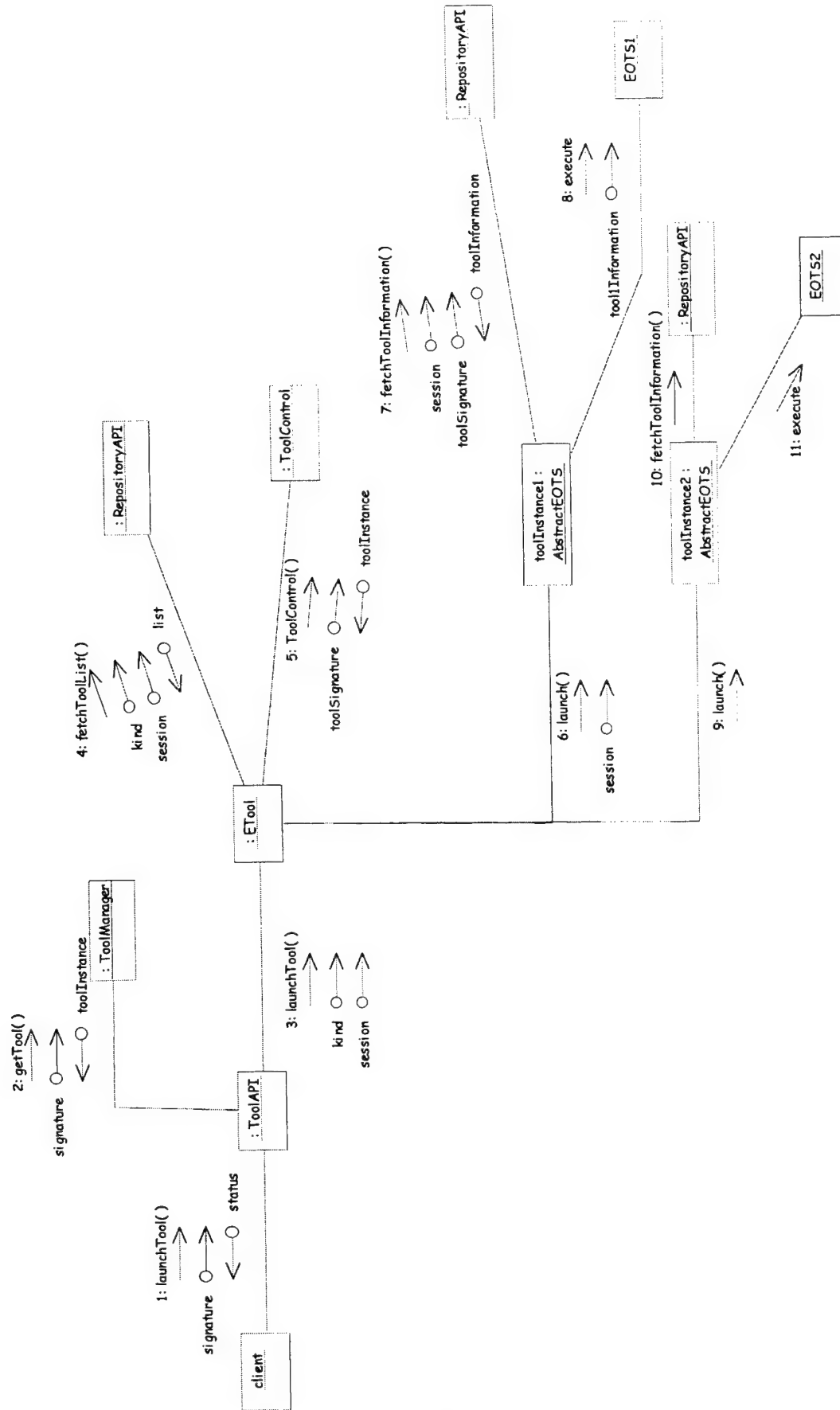


Figure 8-13 - Launch Tool

8.6 Software Design

8.6.1 Packages

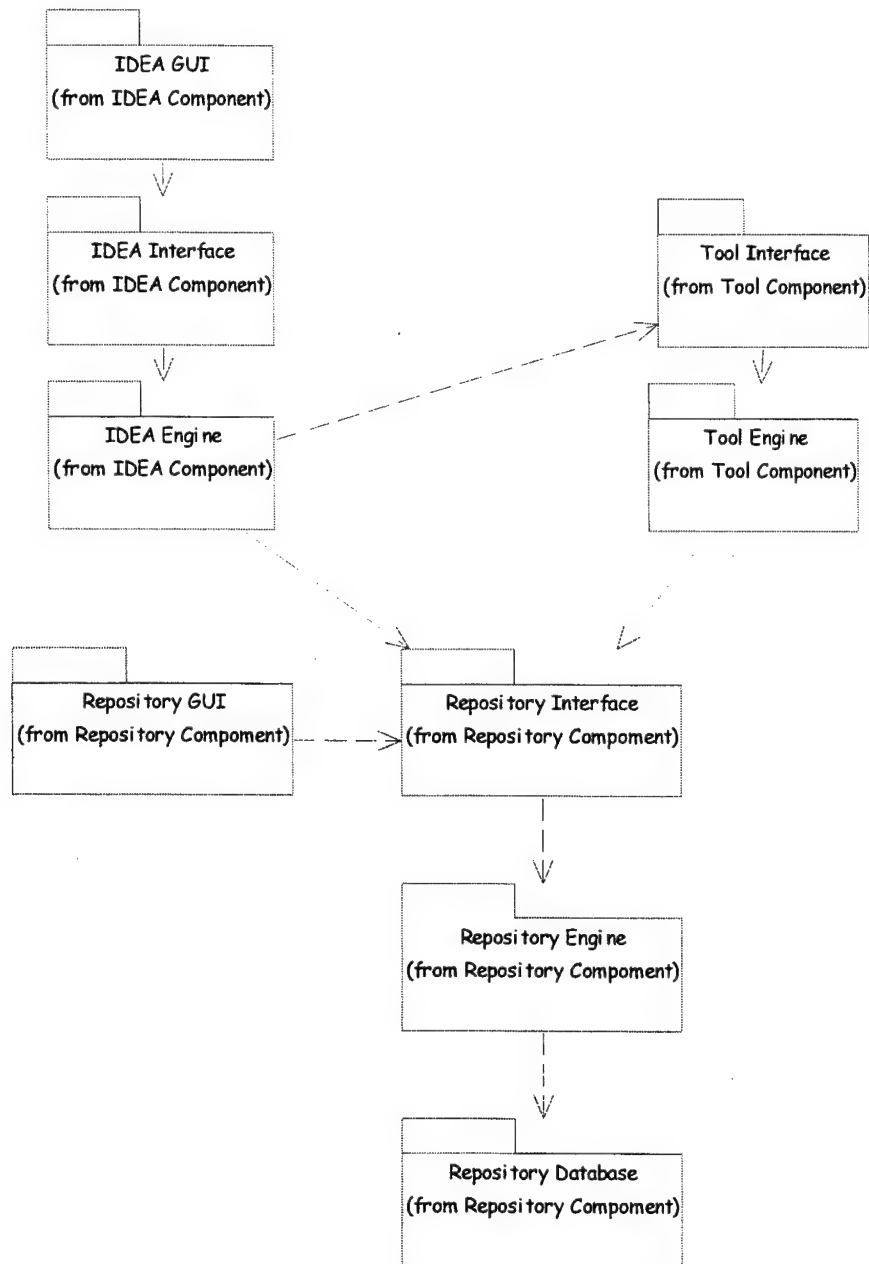


Figure 8-14 - Component Architecture

8.6.2 IDEA Component

The IDEA component is composed of classes supporting user interaction IA design and tool automation. The IDEA Component contains an API within its user interface layer. External calls to IDEA features are supported by the API.

The scope of this component is to route the user's requests to the Tool and Repository components.

The IDEA Component layer is designed to allow users to select and open vulnerability investigations for a network. The mapping capability allows users to set up their network. The populating capability allows users to gather network node information. The analysis capability allows users to analyze the network for vulnerabilities. Fuzzy Fusion analysis is also supported.

8.6.2.1 IDEA API

As stated above, the IDEA API provides a set of operations intended to support vulnerability analysis of a network. This section describes the IDEAAPI class.

8.6.2.1.1 Class Diagram

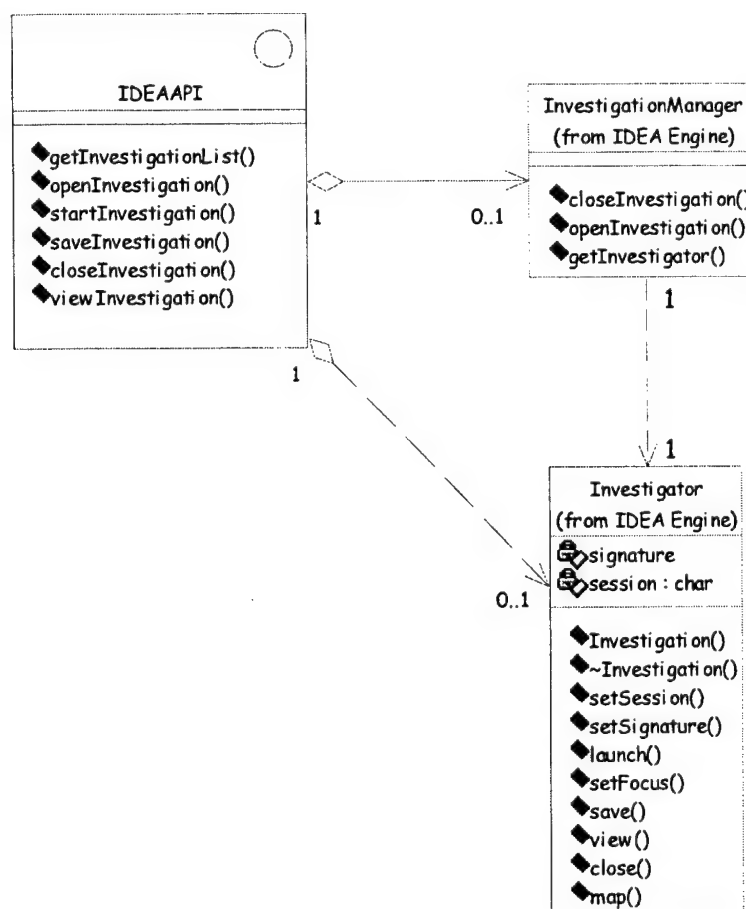


Figure 8-15 - IDEAAPI Class

The IDEAAPI class allows external clients to launch external tools such as discovery, scanning, and analysis tools.

The IDEA API is designed to allow only string or other basic data types past the component boundary. Users are provided a key, or investigation signature to use. This key ensures access to internally maintained investigation information and is used in all IDEA API requests.

8.6.2.1.2 IDEA Error Return Codes

The minimal set of error return codes from the IDEA API are:

IA_SUCCESS

IA_DATABASE_ERROR

IA_INVESTIGATION_NOT_FOUND

IA_INVESTIGATION_INFORMATION_NOT_FOUND

IA_LAUNCH_TOOL_NOT_FOUND

IA_UNABLE_TO_LAUNCH_TOOL

IA_UNABLE_TO_SAVE_INVESTIGATION

IA_INVESTIGATION_NOT_OPEN

IA_INSUFFICIENT_PRIVILEGES

IA_CONFIRMATION_REQUIRED

IA_INSUFFICIENT_PRIVILEGES

8.6.2.1.3 getInvestigationList

NAME

getInvestigationList - Get a list of investigations available to the client.

DESCRIPTION

The **getInvestigationList** command informs the IDEA Engine that a client wishes to select an investigation.

```
IAErrRetType getInvestigationList (  
    in investigationKind kind,  
    out list pSigList)
```

Argument	Description
kind	One of MAP, POPULATE, ANALYZE, FUSE, ALL.
pSigList	Pointer to a string list to place the names of investigations available for the client.

ERROR RETURN VALUE

```

IA_SUCCESS
IA_DATABASE_ERROR
IA_INVESTIGATION_NOT_FOUND
IA_INSUFFICIENT_PRIVILEGES

```

8.6.2.1.4 openInvestigationList

NAME

openInvestigationList - Select, from a list of possible investigations, one to open.

DESCRIPTION

The **openInvestigationList** command informs the IDEA engine that the client is ready to perform operations on the investigation. The mode is used to designate whether the investigation information requires locking so that another client cannot modify the data.

```

IAErrRetType openInvestigationList (
    in  investigationMode mode,
    in  string pName,
    out string pSig)

```

Argument	Description
mode	One of READONLY or WRITE.
pName	Pointer to a string that contains the name of the investigation to open.
pSig	Pointer to a string that contains the key for the investigation. The client must use this key for most interactions with the IDEA API.

ERROR RETURN VALUE

```

IA_SUCCESS
IA_DATABASE_ERROR
IA_INVESTIGATION_INFORMATION_NOT_FOUND
IA_INSUFFICIENT_PRIVILEGES

```

8.6.2.1.5 startInvestigation

NAME

startInvestigation -

DESCRIPTION

The **startInvestigation** command launches the tools on the investigations tool list.

```
IAErrRetType startInvestigation (  
                                in string pName)
```

Argument	Description
pName	Pointer to a string that contains the name of the investigation to start.

ERROR RETURN VALUE

IA_SUCCESS

IA_DATABASE_ERROR

IA_INSUFFICIENT_PRIVILEGES

8.6.2.1.6 saveInvestigation

8.6.2.1.7 closeInvestigation

8.6.2.1.8 viewInvestigation

8.6.2.2 IDEA Engine

The IDEA Engine layer is designed to interface between IDEAAPI clients and the Tool and Repository components. The IDEA Engine layer depends upon the ToolAPI and the RepositoryAPI. Classes contained within this layer support routing client requests to the other components.

Since each session is an object, IDEA is multi-user. The IDEA Component can handle multiple requests from users. We have the potential to setup an IDEA server that handles requests from other users.

8.6.2.2.1 Class Diagram

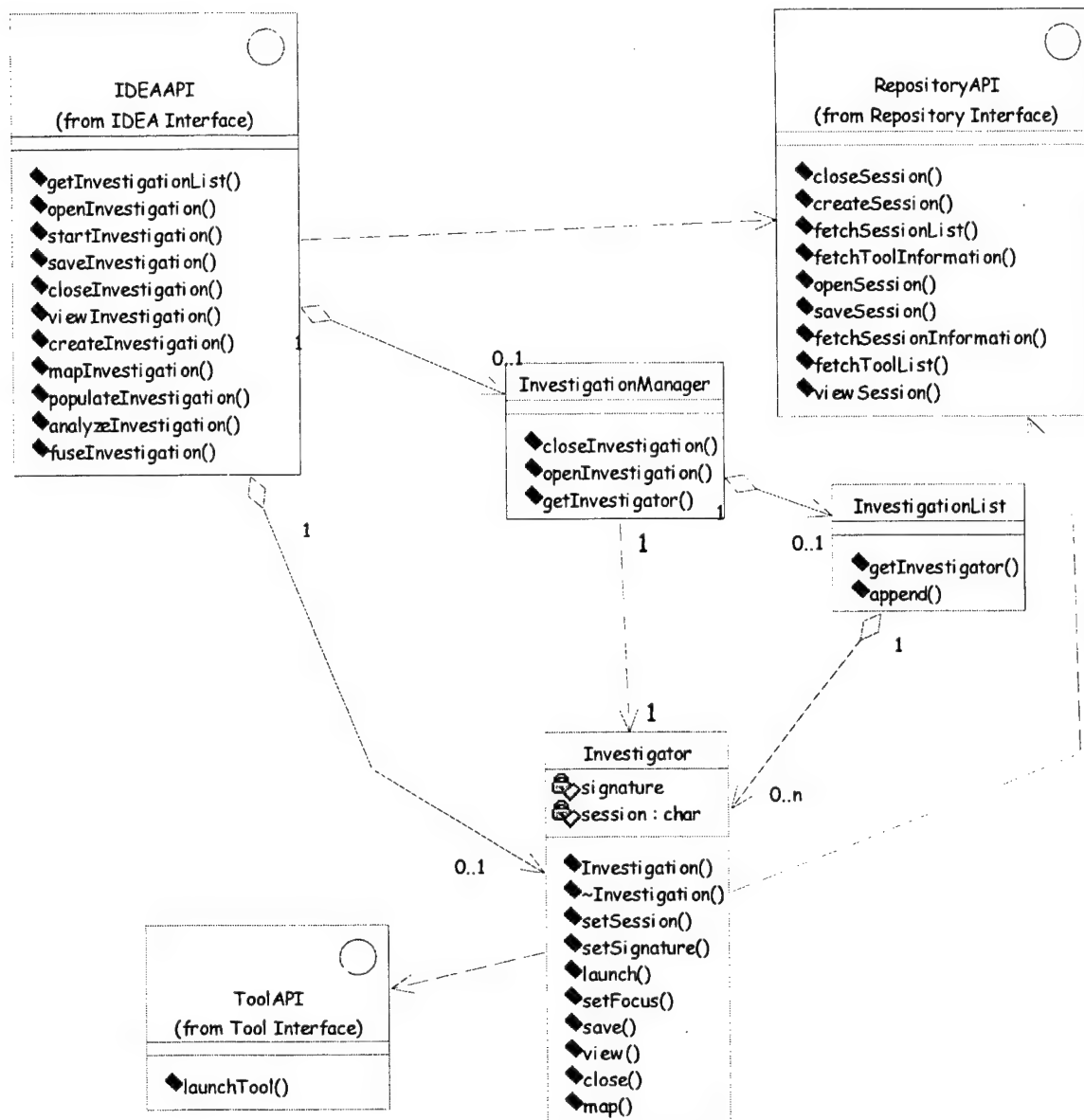


Figure 8-16 - IDEA Engine Layer

8.6.2.2.2 Class Descriptions

InvestigationManager

The role of the InvestigationManager class is to instantiate, store, and destroy Investigator objects. Investigator objects are stored on a Map class - InvestigationList.

Investigator

The role of the Investigator class is to mirror the Repository Session class. Instances of Investigator contain a key representing an instance of Session, as clients to the IDEA API are provided a key representing the investigator instance. IDEA API calls are routed to appropriated Investigator instances. From the instance, the message is routed to the Repository or Tool components by means of their respective API's.

InvestigationList

The role of the InvestigationList is to store investigation names (client readable) and provide dictionary/hash access to them. The STL Map class is used for InvestigationList.

8.6.3 Repository Component

The Repository component is composed of classes supporting database storage and retrieval. The IDEA and Tool component send requests via the Repository API and the classes within the Repository Engine layer interact with the database.

8.6.3.1 Repository API

8.6.3.1.1 Class Diagram

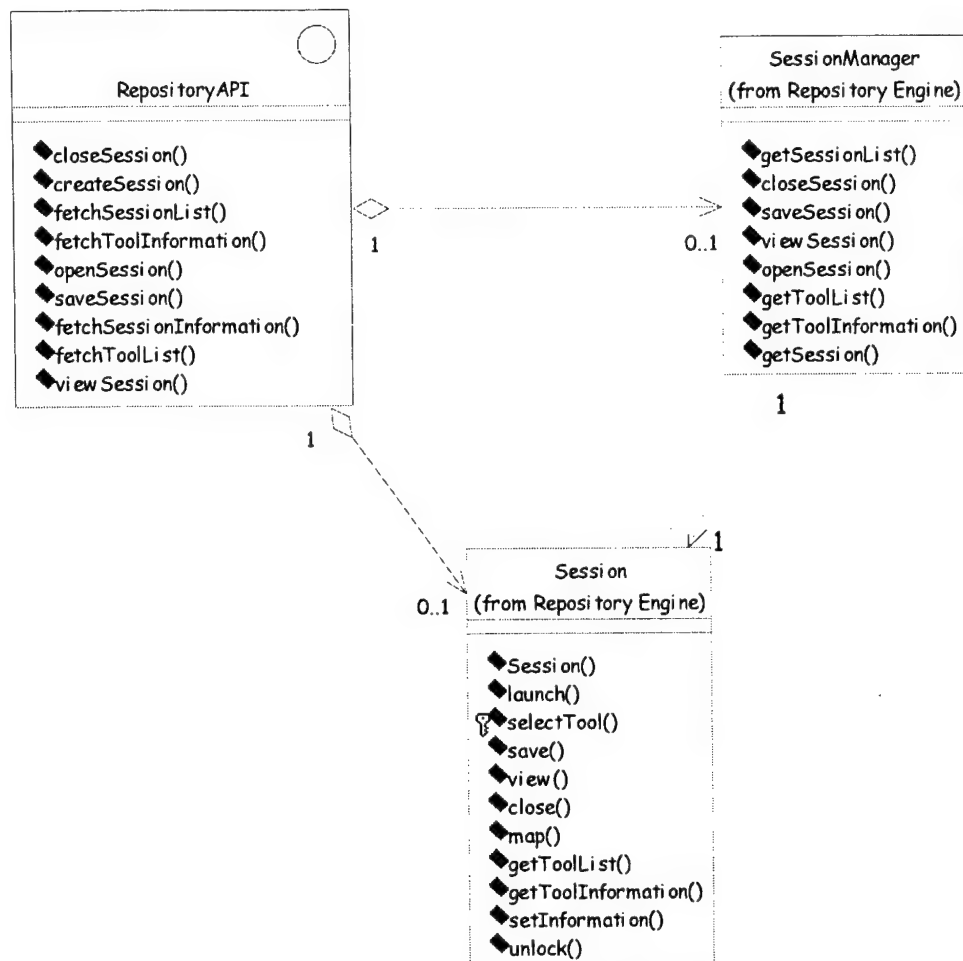


Figure 8-17 - RepositoryAPI

8.6.3.1.2 Repository Error Return Codes

8.6.3.1.3 fetchSessionList

NAME

DESCRIPTION

Argument	Description
----------	-------------

ERROR RETURN VALUE

8.6.3.1.4 openSession

NAME

DESCRIPTION

Argument	Description
-----------------	--------------------

ERROR RETURN VALUE

8.6.3.1.5 fetchSessionInformation

NAME

DESCRIPTION

Argument	Description
-----------------	--------------------

ERROR RETURN VALUE

8.6.3.1.6 saveSession

NAME

DESCRIPTION

Argument	Description
-----------------	--------------------

ERROR RETURN VALUE

8.6.3.1.7 closeSession

NAME

DESCRIPTION

Argument	Description
-----------------	--------------------

ERROR RETURN VALUE

8.6.3.1.8 viewSession

NAME

DESCRIPTION

Argument	Description
-----------------	--------------------

ERROR RETURN VALUE

8.6.3.1.9 createSession

NAME

DESCRIPTION

Argument	Description
-----------------	--------------------

ERROR RETURN VALUE

8.6.3.1.10 fetchToolList

NAME

DESCRIPTION

Argument	Description
-----------------	--------------------

ERROR RETURN VALUE

8.6.3.1.11 fetchToolInformation

NAME

DESCRIPTION

Argument	Description
-----------------	--------------------

ERROR RETURN VALUE

8.6.3.2 Repository Engine

8.6.3.2.1 Class Diagram

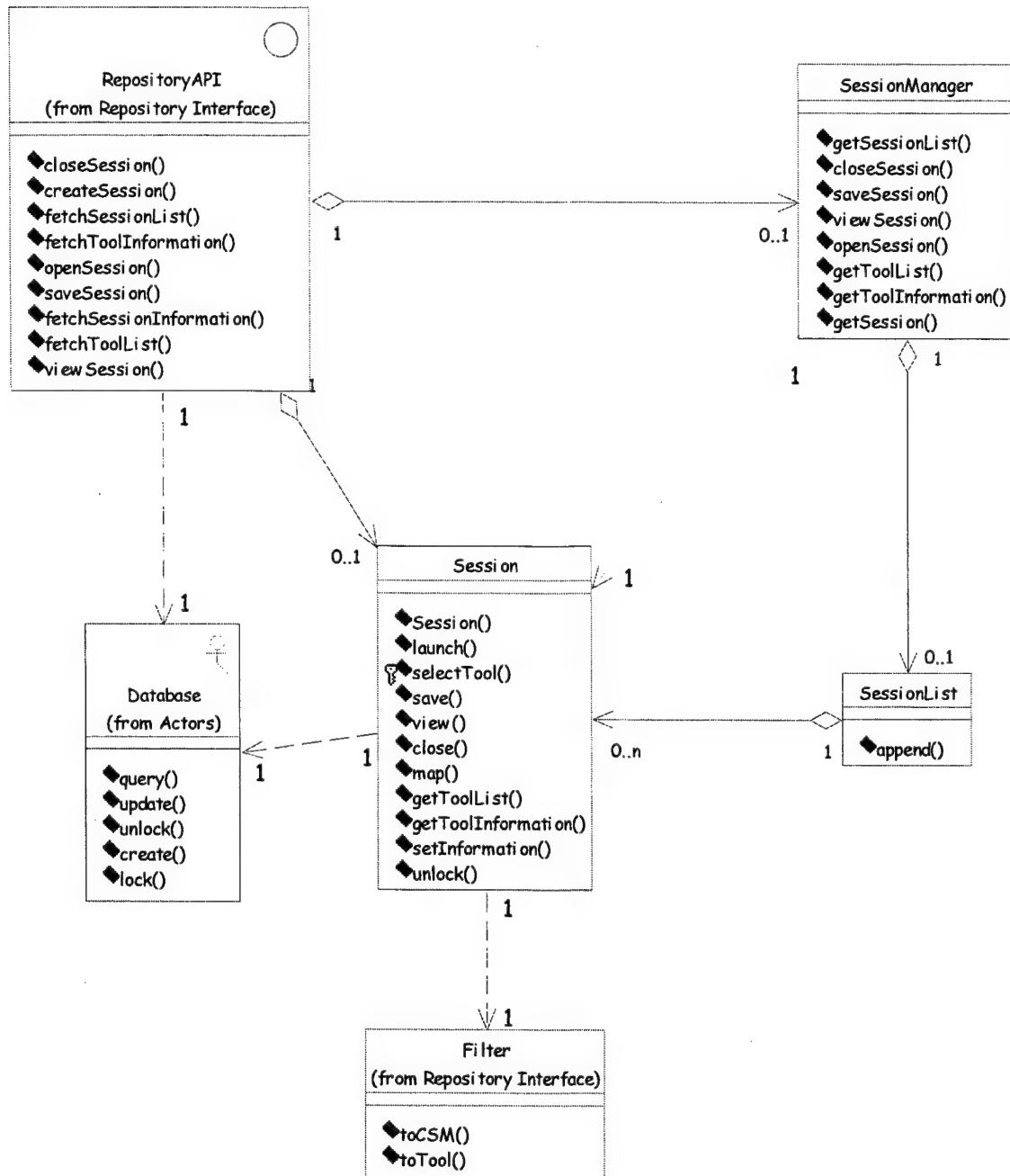


Figure 8-18 - Repository Engine Layer

8.6.3.2.2 Class Descriptions

8.6.3.2.2.1 SessionManager

8.6.3.2.2.2 Session

8.6.3.2.2.3 SessionList

8.6.3.2.2.4 Filter

8.6.3.3 Repository Database

8.6.4 Tool Component

8.6.4.1 Tool API

8.6.4.1.1 Class Diagram

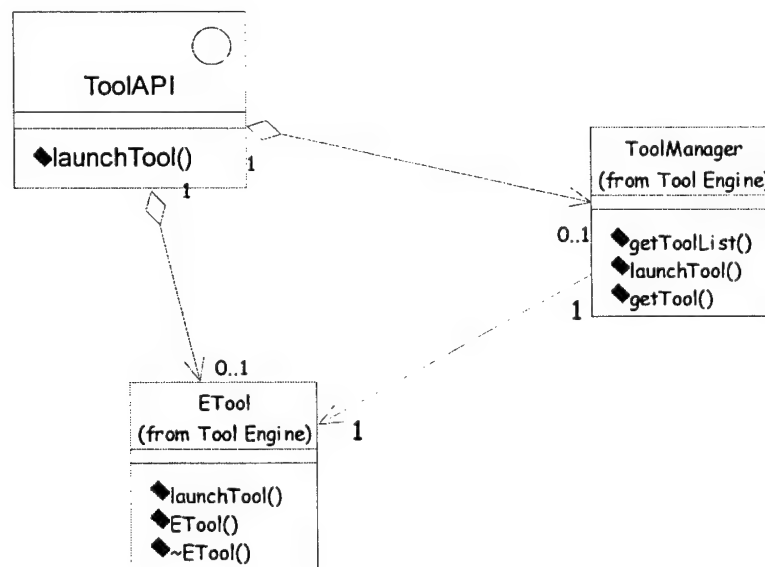


Figure 8-19 - ToolAPI

8.6.4.1.2 Tool Error Return Codes

8.6.4.1.3 launch

NAME

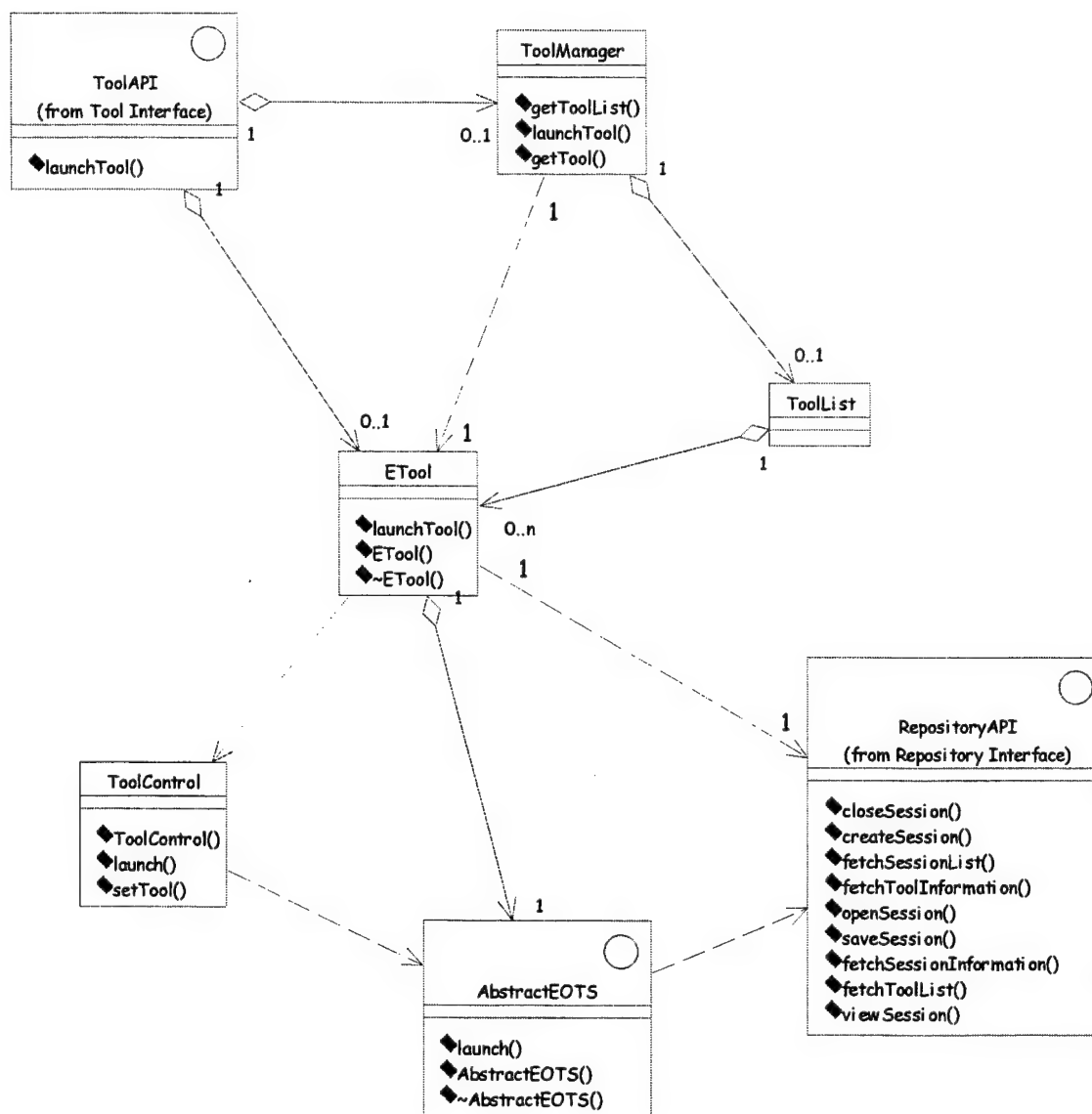
DESCRIPTION

Argument	Description
----------	-------------

ERROR RETURN VALUE

8.6.4.2 Tool Engine

8.6.4.2.1 Class Diagram



8.6.4.2.2 Class Descriptions

ToolManager

ETool

ToolList

ToolControl

AbstractEOTS

9 Deployment View

Note: This section was not completed due to a pre-mature conclusion of the effort.

10 Size and Performance

Note: This section was not completed due to a pre-mature conclusion of the effort.

11 Security

Access to behavior within IDEA is limited to users registered within the database. These are divided into two fundamental categories: IDEA Users, and IDEA Administrators.

IDEA Users include personnel with privileges to map, populate, and analyze systems. The three basic roles performed by IDEA Users are Define Network System, Scan System for Vulnerabilities, and Analyze Vulnerabilities. Administrators will use IDEA to create sessions, define security profiles, and configure tools for sessions.

11.1 Database Security²

Multi-user database systems, such as Oracle, include security features that control how a database is accessed and used. For example, security mechanisms:

- prevent unauthorized database access
- prevent unauthorized access to schema objects
- control disk usage
- control system resource usage (such as CPU time)
- audit user actions

Associated with each database user role is a schema. A schema is a logical collection of database objects (tables, views, sequences, synonyms, indexes, clusters, procedures, functions, packages, and database links).

Database security can be classified into two distinct categories: system security and data security.

System security includes the mechanisms that control the access and use of the database at the system level.

For example, system security includes:

- valid username/password combinations
- the amount of disk space available to a user's schema objects
- the resource limits for a user

System security mechanisms check:

- whether a user is authorized to connect to the database
- whether database auditing is active
- which system operations a user can perform

Data security includes the mechanisms that control the access and use of the database at the schema object level.

For example, data security includes:

- which users have access to a specific schema object and the specific types of actions allowed for each user on the schema object (for example, user SCOTT can issue SELECT and INSERT statements but not DELETE statements using the EMP table)
- the actions, if any, that are audited for each schema object

² Oracle 8, Release 8.0.4 Documentation Library - Oracle8 Concepts Release 8.0 A58227-01

11.2 Security Mechanisms

The Oracle server provides discretionary access control, which is a means of restricting access to information based on privileges. The appropriate privilege must be assigned to a user in order for that user to access a schema object. Appropriately privileged users can grant other users privileges at their discretion; for this reason, this type of security is called "discretionary".

Oracle manages database security using several different facilities:

- database users and schemas
- privileges
- roles
- storage settings and quotas
- resource limits
- auditing

11.2.1 Database Users and Schemas

Each Oracle database has a list of usernames. To access a database, a user must use a database application and attempt a connection with a valid username of the database. Each username has an associated password to prevent unauthorized use. Each user has a security domain - a set of properties that determine such things as the:

- actions (privileges and roles) available to the user
- tablespace quotas (available disk space) for the user
- system resource limits (for example, CPU processing time) for the user

11.2.2 Privileges

A privilege is a right to execute a particular type of SQL statement. Some examples of privileges include the

- right to connect to the database (create a session)
- right to create a table in your schema
- right to select rows from someone else's table
- right to execute someone else's stored procedure

The privileges of an Oracle database can be divided into two distinct categories: system privileges and schema object privileges.

a. System Privileges

System privileges allow users to perform a particular system wide action or a particular action on a particular type of schema object. For example, the privileges to create a tablespace or to delete the rows of any table in the database are system privileges. Many system privileges are available only to administrators and application developers because the privileges are very powerful.

b. Schema Object Privileges

Schema object privileges allow users to perform a particular action on a specific schema object. For example, the privilege to delete rows of a specific table is an object privilege. Object privileges are granted (assigned) to end-users so that they can use a database application to accomplish specific tasks.

Privileges are granted to users so that they can access and modify data in the database. A user can receive a privilege two different ways:

1. Privileges can be granted to users explicitly. For example, the privilege to insert records into the EMP table can be explicitly granted to the user SCOTT.
2. Privileges can be granted to roles (a named group of privileges), and then the role can be granted to one or more users. For example, the privilege to insert records into the EMP table can be granted to the role named CLERK, which in turn can be granted to the users SCOTT and BRIAN.

Because roles allow for easier and better management of privileges, privileges are normally granted to roles and not to specific users. The following section explains more about roles and their use.

11.2.3 Roles

Oracle provides for easy and controlled privilege management through roles. Roles are named groups of related privileges that are granted to users or other roles. The following properties of roles allow for easier privilege management:

- Reduced granting of privileges - Rather than explicitly granting the same set of privileges to many users, a database administrator can grant the privileges for a group of related users granted to a role. And then the database administrator can grant the role to each member of the group.
- Dynamic privilege management - When the privileges of a group must change, only the privileges of the role need to be modified. The security domains of all users granted the group's role automatically reflect the changes made to the role.
- Selective availability of privileges - The roles granted to a user can be selectively enabled (available for use) or disabled (not available for use). This allows specific control of a user's privileges in any given situation.
- Application awareness - A database application can be designed to enable and disable selective roles automatically when a user attempts to use the application.

Database administrators often create roles for a database application. The DBA grants an application role all privileges necessary to run the application. The DBA then grants the application role to other roles or users. An application can have several different roles; each granted a different set of privileges that allow for more or less data access while using the application.

The DBA can create a role with a password to prevent unauthorized use of the privileges granted to the role. Typically, an application is designed so that when it starts, it enables the proper role. As a result, an application user does not need to know the password for an application's role.

11.2.4 Storage Settings and Quotas

Oracle provides means for directing and limiting the use of disk space allocated to the database on a per user basis, including default and temporary tablespaces and tablespace quotas.

a. Default Tablespace

Each user is associated with a default tablespace. When a user creates a table, index, or cluster and no tablespace is specified to physically contain the schema object, the user's default tablespace is used if the user has the privilege to create the schema object and a quota in the specified default tablespace. The default tablespace feature provides Oracle with information to direct space usage in situations where schema object's location is not specified.

b. Temporary Tablespace

Each user has a temporary tablespace. When a user executes a SQL statement that requires the creation of temporary segments (such as the creation of an index), the user's temporary tablespace is used. By directing all users' temporary segments to a separate tablespace, the temporary tablespace feature can reduce I/O contention among temporary segments and other types of segments.

c. **Tablespace Quotas**

Oracle can limit the collective amount of disk space available to the objects in a schema. Quotas (space limits) can be set for each tablespace available to a user. The tablespace quota security feature permits selective control over the amount of disk space that can be consumed by the objects of specific schemas.

11.2.5 Profiles and Resource Limits

Each user is assigned a profile that specifies limitations on several system resources available to the user, including:

- a. number of concurrent sessions the user can establish
- b. CPU processing time
 - 1. available to the user's session
 - 2. available to a single call to Oracle made by a SQL statement
- c. amount of logical I/O
 - 1. available to the user's session
 - 2. available to a single call to Oracle made by a SQL statement
- d. amount of idle time for the user's session allowed
- e. amount of connect time for the user's session allowed
- f. password restrictions
 - 1. account locking after multiple unsuccessful login attempts
 - 2. password expiration and grace period
 - 3. password reuse and complexity restrictions

Different profiles can be created and assigned individually to each user of the database. A default profile is present for all users not explicitly assigned a profile.

The resource limit feature prevents excessive consumption of global database system resources.

11.2.6 Auditing

Oracle permits selective auditing (recorded monitoring) of user actions to aid in the investigation of suspicious database use. Auditing can be performed at three different levels: statement auditing, privilege auditing, and schema object auditing.

a. **Statement Auditing**

Statement auditing is the auditing of specific SQL statements without regard to specifically named schema objects. In addition, database triggers allow a DBA to extend and customize Oracle's built-in auditing features. Statement auditing can be broad and audit all users of the system or can be focused to audit only selected users of the system. For example, statement auditing by user can audit connections to and disconnections from the database by the users SCOTT and LORI.

b. **Privilege Auditing**

Privilege auditing is the auditing of the use of powerful system privileges without regard to specifically named schema objects. Privilege auditing can be broad and audit all users or can be focused to audit only selected users.

c. **Schema Object Auditing**

Schema object auditing is the auditing of accesses to specific schema objects without regard to user. Object auditing monitors the statements permitted by object privileges, such as SELECT or DELETE statements on a given table.

For all types of auditing, Oracle allows the selective auditing of successful statement executions, unsuccessful statement executions, or both. This allows monitoring of suspicious statements, regardless of whether the user issuing a statement has the appropriate privileges to issue the statement.

The results of audited operations are recorded in a table referred to as the audit trail. Predefined views of the audit trail are available so that you can easily retrieve audit records.

11.3 Authenticating Database Users with Windows NT³

11.3.1 Authentication Overview

The Oracle8 database can use information maintained by Windows NT to authenticate database users. The benefits of Windows NT authentication include:

- Enabling users to connect to an Oracle8 database without supplying a user name or password
- Centralizing Oracle8 database user authorization information in Windows NT, which frees Oracle8 from storing or managing user passwords
- Allowing Oracle8 and Windows NT user names to be the same

The Windows NT Native Authentication Adapter (automatically installed with Net8 Server and Net8 Client) enables database user authentication through Windows NT. This enables client computers to make secure connections to an Oracle8 database. A secure connection is when a Windows NT client user name is retrieved on a database server through the Windows NT Native Authentication Adapter. The database server then permits the user name to perform the database actions on the server.

The Windows NT Native Authentication Adapter provides database users with the following privileges:

- Connecting Without a Password as a Nonprivileged Database User
- Connecting as SYSOPER and SYSDBA Without a Password
- Connecting as INTERNAL Without a Password
- Granting Database Roles through Windows NT

Attention:

Granting database roles through Windows NT is an advanced database administration task not appropriate for all database environments. You cannot use both Windows NT and the Oracle8 database to grant roles concurrently. Select a role granting process appropriate to your database environment.

Note:

For Windows NT authentication to work, the SQLNET.AUTHENTICATION_SERVICES parameter must be set as follows in your *ORACLE_HOME*\NET80\ADMIN\SQLNET.ORA file on both client and server:

³ Oracle8 Enterprise Edition – Getting Started

SQLNET.AUTHENTICATION_SERVICES = (NTS)

This is the default setting after Net8 Server and Net8 Client installation.

11.3.2 Connecting Without a Password as a Nonprivileged Database User

This section describes how to authenticate nonprivileged database users (nondatabase administrators) using Windows NT so that a password is not required when accessing the database. When you use Windows NT to authenticate nonprivileged database users, your database relies solely on Windows NT to restrict access to database user names.

The local and domain user name FRANK and the domain SALES are used in the steps below. Substitute the appropriate local and domain user name and domain name for your environment.

To perform authentication tasks on an Oracle8 database server:

- a. Add the OS_AUTHENT_PREFIX parameter to your INITSID.ORA file. The OS_AUTHENT_PREFIX value is prefixed to local or domain user names attempting to connect to the server with the user's operating system name and password. The prefixed user name is compared with the Oracle user names in the database when a connection request is attempted.
- b. Set OS_AUTHENT_PREFIX to "":
- c. Use User Manager to create a Windows NT domain user name for FRANK (if the appropriate name does not currently exist). See your Windows NT documentation or your network administrator if you do not know how to do this.
- d. Follow the substeps below to create a new registry parameter for authenticating a domain name FRANK on domain SALES.
 1. Start the registry editor, REGEDIT32, from Start/Run:
Go to the registry subkey of the Oracle home directory that you are using.
 2. Choose the Add Value option in the Edit menu.
 3. Enter OSAUTH_PREFIX_DOMAIN in the Value Name field.
 4. Choose REG_EXPAND_SZ from the Data Type drop-down list box.
 5. Click OK.
The *String Editor* dialog box appears:
 6. Enter TRUE in the String field to enable authentication at the domain level. TRUE enables the server to differentiate between multiple FRANK user names, whether they be local user FRANK, domain user FRANK on SALES, or domain user FRANK on another domain in your network. Entering FALSE causes the domain to be ignored and local user FRANK to become the default value of the operating system user returned to the server.
 7. Click OK.
The Registry Editor adds the parameter.
 8. Choose Exit from the registry menu.
- e. Ensure that SQLNET.AUTHENTICATION_SERVICES is set as follows in your ORACLE_HOME\NET80\ADMIN\SQLNET.ORA file:

SQLNET.AUTHENTICATION_SERVICES = (NTS)

- f. Start Server Manager:
SVRMGR30
- g. Connect to the database with the SYSTEM database administrator (DBA) name:
SVRMGR> CONNECT SYSTEM/PASSWORD
- h. Create an operating system-authenticated user by entering the following:
SVRMGR> CREATE USER "SALES\FRANK" IDENTIFIED EXTERNALLY;
The double quotes are required and the entire syntax *must* be in uppercase.
- i. Grant the Windows NT domain user FRANK appropriate database roles from the possible choices below:
SVRMGR> GRANT PLANNER TO "SALES\FRANK";
- j. Connect to the database with the INTERNAL DBA name:
SVRMGR> CONNECT INTERNAL/PASSWORD
- k. Shut down the database:
SVRMGR> SHUTDOWN
- l. Restart the database:
SVRMGR> STARTUP
- m. To check enter:
SVRMGR> SELECT * FROM USER_ROLE_PRIVS;

This causes the change to the OS_AUTHENT_PREFIX parameter value to take effect.

11.3.3 Granting Database Roles through Windows NT

This section describes how to grant Oracle8 database roles to users directly through Windows NT. When you use Windows NT to authenticate users, Windows NT local groups can grant these users database roles. Through User Manager, you can create, grant, or revoke database roles to users.

Note:

All privileges for these roles are active when the user connects. When using operating system roles, all roles are granted and managed through the operating system. You cannot use both operating system roles and Oracle roles at the same time.

To perform authentication tasks on the Oracle8 database server:

- a. Add the OS_ROLES initialization parameter to the INITSID.ORA file.
- b. Set OS_ROLES to TRUE.
The default setting for this parameter is FALSE.
- c. Ensure that SQLNET.AUTHENTICATION_SERVICES is set as follows in your ORACLE_HOME\NET80\ADMIN\SQLNET.ORA file:

SQLNET.AUTHENTICATION_SERVICES = (NTS)

- d. Create a new database role:

SVRMGR> CREATE ROLE DBSALES3 IDENTIFIED EXTERNALLY; where DBSALES3 is the name of the role for these steps. Substitute a role name appropriate to your database environment.

- e. Grant Oracle roles to DBSALES3 that are appropriate to your database environment:

SVRMGR> GRANT DBA TO DBSALES3 WITH ADMIN OPTION;

SVRMGR> GRANT RESOURCE TO DBSALES3 WITH ADMIN OPTION;

SVRMGR> GRANT CONNECT TO DBSALES3 WITH ADMIN OPTION;

- f. Connect to the database with the INTERNAL DBA name:

SVRMGR> CONNECT INTERNAL/PASSWORD

- g. Shut down the database:

SVRMGR> SHUTDOWN

- h. Restart the database:

SVRMGR> STARTUP

- i. Open the Windows NT User Manager.

- j. Choose New Local Group from the User menu.

The *New Local Group* dialog box appears:

- k. Enter the Windows NT local group name corresponding to the database role in the Group Name field with the following syntax:

ORA_SID_ROLENAME [_D] [_A]

For this example, ORA_ORCL_DBSALES3_D is entered.

- l. Click *Add*.

The *Add Users and Groups* dialog box appears:

- m. Select the appropriate Windows NT local or domain user name and click *Add*.

- n. Click *OK*.

Your selection is added to the Members field of the *New Local Group* dialog box:

You can convert additional database roles to several possible Windows NT groups, as shown in the following table. Then, users connecting to the ORCL instance in this example and authenticated by Windows NT as members of these Windows NT local groups have the privileges associated with DBSALES3 and DBSALES4 by default (because of the *_D* option). DBSALES1 and DBSALES2 are available for use by the user if they first connect as members of DBSALES3 or DBSALES4 and use the SET ROLE command. If a user tries to connect with DBSALES1 or DBSALES2_A without first connecting with a default role, they are unable to connect. Additionally, users can grant DBSALES2 and DBSALES4 to other roles.

Database Roles	Windows NT Groups
DBSALES1	ORA_ORCL_DBSALES1
DBSALES2	ORA_ORCL_DBSALES2_A
DBSALES3	ORA_ORCL_DBSALES3_D
DBSALES4	ORA_ORCL_DBSALES4_DA

Note:

When the Oracle8 database converts the group name to a role name, it changes the name to uppercase.

- o. Click OK.
- p. Exit User Manager.

12 Appendix A

12.1 *CSM minimum information*

Define the minimum set of information needed to be put into the CSM to launch each tool

- 1) Detail where the information can come from. Will it come from a user or some other tool?
- 2) Define the optimal (most minimum) path for executing the tools. Take into account all configured tools - this means we need a list of configured tools available/accessible. Perhaps take into account all known tools - this would be useful to let the user know that if they purchase this other tool, it would make their life easier.

12.2 *Taxonomies*

Table - Layer Taxonomy

Table - Node Taxonomy

Table - Attribute Taxonomy

13 Appendix B

13.1 *Table of CSM Layers*

This table shows:

- the contents of each layer in terms of nodes,
- the attributes available for each node itemized by layer, and
- enumerates possible relationships to other layers.

14 Appendix C

14.1 Table of IA Tool API Capabilities

This table shows:

- the tool platform,
- the protocol required to launch the tool,
- the communication backplane used to launch the tool,
- data format translation from CSM for the tool,
- the location of the tool results,
- the format translation from the Tool results to CSM,
- what tool results are valid for the CSM,
- what tool results are not applicable to the CSM, and
- defines what to do with the non-CSM tool results.

Table 14-1 - Tool configuration

Tool Name	
Tool Supporters	What other tools does this tool require data from
Tool Clients	What other tools require this tool's information
Tool Map	Mapping of the CSM schema to the tool's required data
Tool Results	Map of the tool's results to the CSM schema

15 Glossary

Common System Model (CSM) - The Common System Model is a schema for defining and storing network and other system related information.

Map - The use of discovery tools or network design tools is termed *Mapping*.

Populate - The use of scanning tools is called *Populating*. A *Map* is *Populated* through the use of scan tools.

Analyze - The use of analysis tools (with the exception of Fuzzy Fusion) is called *Analyze*. A *Populated Map* is *Analyzed* for vulnerabilities.

Fuse - The use of the Fuzzy Fusion tool is termed *Fusing*. Vulnerabilities and other system qualities found during the *Analysis* of a *Populated Map* are *Fused*.

Model - A model is a snapshot of the system network, hardware, and software. Model information is used by external tools during their system evaluation. The model information is stored as part of a session.

Session - A session is an entry in the repository for the collection of network information, tool results, and security profile information.

Investigation - The term investigation is used to denote the act of mapping, populating, analyzing, and fusing a network.

Appendix E

Meta-Language

Information Fusion Meta-Language Requirements

Name	Ronda R. Henning	Margaret Knepper	Kevin L. Fox
Address	Harris Corporation Government Communications Systems Division P.O. Box 37 Mail Stop W2/9703 Palm Bay, FL 32905	Harris Corporation Government Communications Systems Division P.O. Box 37 Mail Stop 2/9450 Palm Bay, FL 32905	Harris Corporation Government Communications Systems Division P.O. Box 37 Mail Stop 2/9450 Palm Bay, FL 32905
Phone Number	321-984-6009	321-727-5268	321-729-7119
E-Mail	rhenning@harris.com	mknepper@harris.com	kfox@harris.com

Introduction

Most security language related work has focused on mathematically correct security policy expressions that can be formally verified with theorem proving techniques [1] [2]. Our work on the IDEA program [3] focused on the definition of an information assurance superstructure that could facilitate data sharing among information assurance tools. In the course of our work on IDEA, it became evident that a security translation function mapping the end-user's interpretation of security domain terminology to the more precise terminology of the security engineer was required. During the design process it became increasingly apparent that the specificity and precision needed to translate user security statements into effective, reasonably secure implementations was lacking. A mechanism to facilitate requirement solicitation and translation between the end user and security development domains was also required. To accomplish these goals, the use of a meta-language was proposed to develop the correspondence functions. This paper describes our efforts in designing the meta-language structure, and our progress to date on its implementation.

Meta-Languages

A *meta-language* is a language used to describe other languages [3]. Meta-Languages are applicable to and used in several different areas of computer science:

- Meta-Language for IP – A meta-language for the transformational programming environment [4].
- WebSite META Language - A sequential filtering scheme where each language provides one of nine processing passes [5].
- Financial Metal-Language - Provides a dialect specific to the financial industry for the description of both data and processing [6].

Within the context of information assurance and information operations, a meta-language allows designers and analysts to express the following inherent properties of a system:

- the relationships between system elements,
- the causalities, or which elements are relationships are the cause of system actions,
- the vulnerabilities, or the potential opportunities for system security exploitation
- the threats, or the actual opportunities a system user may have to exploit the system's vulnerabilities, and
- the objectives, or mission to be accomplished by a given system.

A security analyst expresses a security policy using a semantically correct syntax that can be readily traced through the design process to an implementation among several security mechanisms. An end-user expresses security in terms of data sharing or access rights within his frame of reference. Our objective was

to define an information assurance translation mechanism that could capture the end user's assurance requirements and facilitate their implementation in and traceability to a security design for a given end user's application domain.

Knowledge Engineering

To constrain the scope of the meta-language development activity, it was necessary to define which aspects of security engineering the meta-language would be able to address most effectively. The Security Engineer rarely builds the same system architecture twice, but usually has the same objectives to embody in each system's design: enforcement of the customer's security policy and the maintenance of the system in a known, secure state. The Security Engineering (SE) discipline applies a variety of information sources and tools to analyze the diverse aspects of system security problems. For example, the physical, electronic, and personnel security requirements of the customer must all be addressed within a system design. The cost and performance impact associated with a given solution set may vary greatly, depending upon which countermeasures are applied.

We must understand the types of information and tools used by the security engineer in order to define a meta-language for the security-engineering domain. To develop this domain understanding, knowledge engineering activities were conducted with experienced system security architects. The objective of our knowledge engineering task was to determine what kinds of information the security engineer needs and how that information is applied in the systems development process.

A system architecture scenario was generated and used in a series of knowledge engineering interviews with multiple system security engineers. Given a relatively "normal" network configuration, the security engineer was asked to evaluate the configuration with the question: "What are the vulnerabilities that allow a person to break into this system?" From this single question we began to characterize the security thought processes and categorize aspects of the security domain. It soon became apparent that the scenario needed to be constrained, or the many aspects of security involved would present an overwhelming number of alternative solutions. For example, physical security, personnel security, network security, and information security all represent valid security aspects of a given system design. For the purposes of the knowledge engineering scenario, we concentrated on information security considerations.

During the knowledge engineering scenario, it became apparent that the security engineer applies an analytical problem solving process to define potential security problems and possible solutions. This process can be characterized in the following steps:

- Understanding the problem, or the system scenario
 - Defining the goal of security in the context of the scenario
 - What information is to be protected?
 - Why does this information require protection?
 - What is it being protected against?
 - How is the information being used?
 - Describing the physical environment in which the system resides.
 - Describing the personnel security practices in place.
 - Describing any network interfaces or connectivity requirements.
- Decomposition of the problem. Based on the understanding gained in step one, the security engineer decomposes the problem into a series of sub-problems. These sub-problems are similar to the steps described in the Anatomy of a Hack [6], which describes the thought process of a system intruder. The sub-problems can be decomposed into the following categories:
 - Access To – How can someone access the system?
 - Access Into – How can someone get access into the system?
 - Access What/how – Once they have access into the system, what can they access and how will they access it?
 - Hide Access – How will they hide their access into the system?

- Resolution of the decomposed problem. Often solving the decomposed sub-problems proved easier than solving the overall scenario. The security engineer addressed each issue exposed in the problem decomposition and, upon solving each individually, solved the security issues for the system as a whole.

Security Meta-Language

Based on the knowledge acquisition process mentioned above, we began to define the information characteristics required for a security specific meta-language. In addition to the meta-language syntax, a set of language interactions must be specified in the meta-language. The meta-language requires three principle components:

1. Definitions – A syntax to describe the security structures and the environment.
2. Procedures – The ability to combine definition objects that can interact with automated tools to facilitate problem analysis.
3. Tool Interaction – The ability to interact among various security tools, providing input to the tools and using the output results to improve the security solution.

To illustrate the structure of the meta-language we present an example using the steps developed in the Knowledge Engineering scenario – “What are the vulnerabilities that allow a person to break into our computer system?”

Definitions

The first step of the process is to define the information required for the analysis. The meta-language structure allows the security engineer to describe the security structures and the environment. The definitions need to be decomposed to the smallest possible components, which can then be recombined to build higher level definitions. Basic operators (AND, OR, NOT etc.) will allow the operator to perform the building of the definitions. The syntax should be defined in terms that the user community understands. A security expert might say ‘discretionary access control;’ another user could define it as ‘read access to the accounting files and nothing else’. Each of these language syntax definitions describes the same concept, but they are able to define it with terminology that is comprehensible to them. The next level of language definition would be to map similar terms between the end user domain and the security-engineering domain.

One of the first things the security engineer defines is the system environment. Instead of repeatedly describing all the information about the environment, it was easier to develop standard descriptions of different environments and allow the user to select the best standard description. Table 1 presents the standard set of environmental descriptions we applied during the knowledge engineering scenarios. The terms firewall, encryption, and EMSEC will need their own definitions to be useful in the meta-language in the long term.

Table 1. Descriptions of the environment				
Environment Type	Open	Restricted Protected Connections	Restricted Isolated Domain	Self Contained
Condition Description	Unprotected external connections AND Physical access to system	Control to physical environment AND [firewall OR encryption to outside] AND Limited EMSEC	Control to physical environment AND [DMZ OR server physically separated] AND Limited EMSEC	No external connections AND Personnel checked before access to system AND No emanation

Based on the environment type selected, acceptable descriptions of Identification and Authentication (I&A) information can be linked with a given selection. Definitions for Identification and Authentication are presented in Table 2. The next level of abstraction would be to link the environment and the security policy of the system to define the appropriate type of Identification and Authentication required.

Table 2. Example of Language Syntax for Identification and Authentication			
Definition	Description	Type	Definition
Identification & Authentication	<ul style="list-style-type: none"> Who's allowed to login How does the system decide who is legitimate How does the system keep track of who's doing what in the system 		
		Basic	Protected mechanism for authentication AND Unique user identification AND All auditable events AND I&A protected
		Enhanced	Basic I&A AND [additional authentication mechanisms OR independent authentication mechanisms OR user access profile OR authentication administration]

The user also needs to define information about the different vulnerabilities associated with the problem. For example, in this scenario, a vulnerability would be external connectivity. Table 3 illustrates the basic definition capability used in the meta-language to define the components of 'External Connectivity'. The components of the 'External Connections TO' category are Internet, dialup, wireless, and network connections. The user can also assign a security level, or priority, in conjunction with the definition. This priority can be treated as a ranking order of potential concerns. In addition to connecting to the network, another vulnerability may be listening to the network, or eavesdropping with a network sniffer device. The definitions for 'External Connection LISTEN' need to be expanded in a manner similar to the 'External Connect TO' definitions.

Table 3. Example of Language Syntax for External Connections					
Definition	Description	Type	Security Level	Protection	Protection Exploitation
External Connection TO	External connections to the system				
		Internet	1		
				None	Easy -
				Encryption < 2	Easy +
				Encryption > 2	Mod -
				Firewall	Mod
		Dialup	2		
		Wireless	3		
		Network	4		
External Connections LISTEN	External connections that can let the user listen to the system				
		Cell/FM			

For example, the Internet connection is the user's highest level of concern. As we walked through the refinement process we realized that not only are there different priorities associated with different threats, there are also different levels or priorities associated with various countermeasures. There are different levels of protection that can be offered, so that information should also be included in the definition. This information can also be used to provide assistance to another user by describing how to protect the system. Finally, the degree of difficulty in penetrating the system must be represented (i.e. how difficult it would be to break into the system). We define this as the possibility of Protection Exploitation, and the value is based upon the protection mechanism employed. The protection exploitation shown in Table 3 is a measure of the difficulty to break into a given system.

Table 4 shows the definition of the protection exploitation, based on the cost, time, and accountability factors associated with penetrating a system. Additionally the '+' and '-' measures were used to further express the relative measurement, with the '+' indicating a higher degree of difficulty. Once we developed this measure it was easy to rate different vulnerabilities. Table 5 shows an example of rating the wireless protections to breaking into the system.

Table 4. Measure of Difficulty of breaking into a system			
Rating	Easy	Moderate	Hard
Cost	< \$1000	< \$10,000	> \$ 10,000
Time	1 hour	< week	> week
Accountability	None	Detection	Detected and recorded
Definition	Little or no effort to break into the system	Some effort to break into the system	A lot of effort to break into the system.

Table 5. Example of Protection Exploitation for Wireless External Connections					
	Cell/FM	Microwave	Light	AM/FM	Burst
None	M-	M	M+	M-	M+
Encryption < 2	M+	H-	H-	M+	H
Encryption > 2	H	H	H	H-	H+

As the process continues, the user continues to develop and refine the required definitions. While walking through our scenario we developed definitions for the different parts of the problem. These definitions included:

- **Access to the system** – The two primary methods of breaking into the system were identified as physical access and the external connectivity to the system. The main emphasis was identifying the vulnerabilities that would permit someone into the system.
- **Access into the system** – We focused on the basic rules of the Identification and Authentication (presented Table 3).
- **Access What** – Data files and system files were the primary items to attack in our environment examples. Files can be read, written, deleted, and executed, depending upon the user's access privileges.
- **Access How** –Data can be accessed through root access, root privilege, change privilege, execute commands, spoofing, system configuration, buffer overflow, or access to files. The main protection against data access was Discretionary Access Control (DAC).
- **Hiding access** – We started the identification process by asking the basic questions: What is recorded? How is it recorded? Where is it stored? We identified the system and its auditing subsystem as the two primary places a user would attack. System data files can be modified or executed. The Auditing file has a number of methods that can be used in an attack, listed in Table 6.

Table 6. Auditing File Protection Methods	
Vulnerability	Protection
Turn it off	System Administration
	Access Control
Delete it	Access Control
Overwrite it	System Configuration
Reset it	Permission
Drop Bit Bucket	Permission

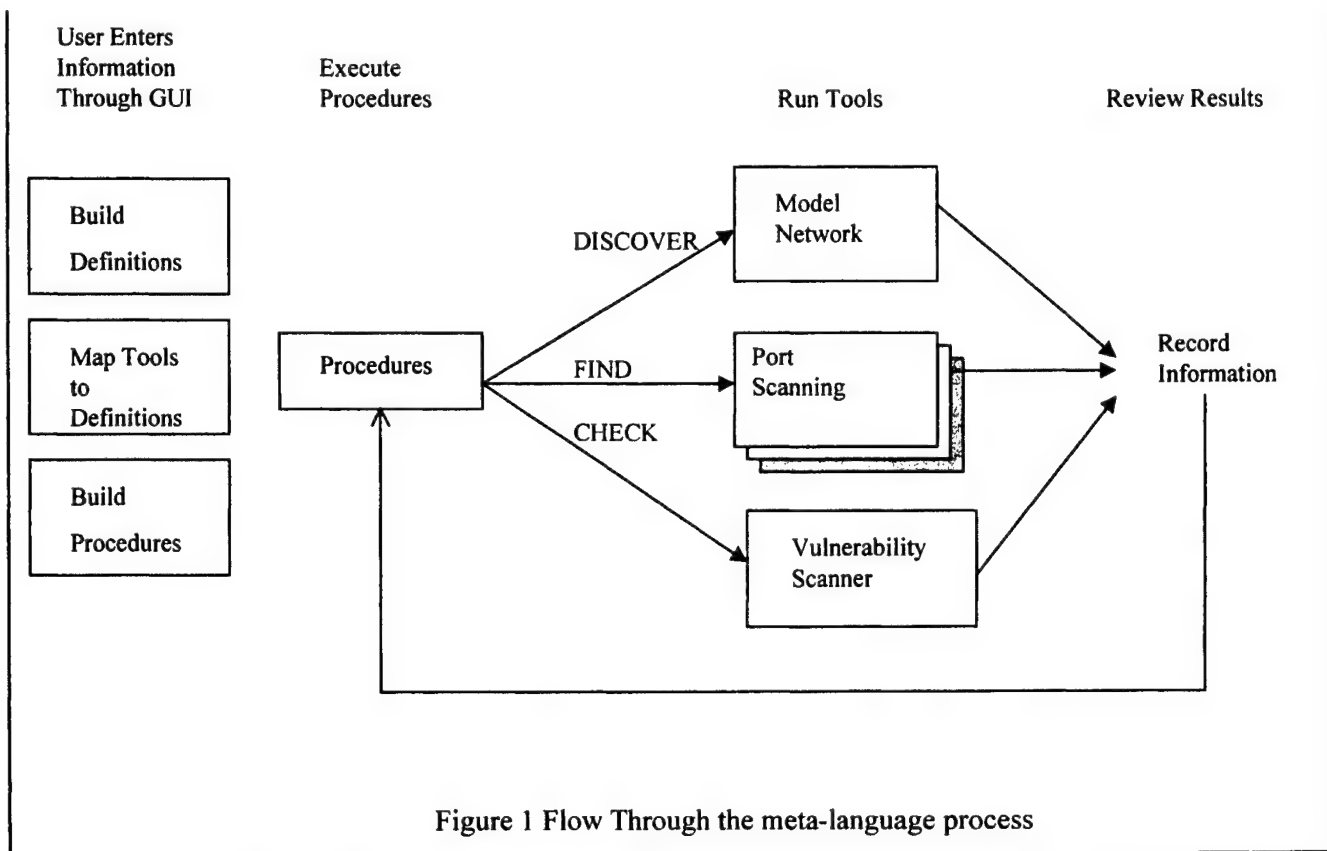


Figure 1 Flow Through the meta-language process

Integration with Security Tools

The definitions are mapped to each security tool's inputs and outputs. The meta-language accommodates special directives for direct interaction with the security tools. The user builds a procedure that combines a set of directives to be executed, assisting the security engineer in problem analysis. In this scenario the user may want to perform the following steps to help them analyze the problem:

SET environment = Restricted Protected Connections.
 DISCOVER the network.
 FIND external connections.
 CHECK the Identification and Authentication.

These steps refer to specific actions defined in the context of specific tools, or the specification of particular environmental constraints. Once the procedures are in place, then the meta-language can be executed, as shown in Figure 1. After the procedure has executed, the SE is able to review the results.

Language Issues

As we built the meta-language syntax, the need for a reasoning capability became apparent. In the case of the Identification and Authentication definition, suppose one of the pieces of the definition is not being implemented. How should the protection level be rated when piece(s) of the definition are missing? Does the user have to examine each possible scenario for missing information and rate it? This would be a very unpleasant and time-consuming process for the user. The security engineer is able to analyze the problem and determine that the loss of one element of Identification and Authentication may not be too severe due to environmental considerations and other protections in the system. We do not believe this thought process should be considered a part of the meta-language. Our current thought is that this process requires another tool to perform this reasoning process. The meta-language will funnel information into the reasoning capability tool and provide analyzed results.

Progress to Date

The language syntax definitions need to be tested with other system architecture scenarios to validate and expand the language's vocabulary. The language syntax needs to be sufficiently extensive to describe the

various layers of information detail that will be required for a complete system description, and also needs to have enough flexibility to evolve as system architectures and countermeasures evolve over time.

Most of our work on IDEA work has focused on the security architecture for the system and its derivation. Our work on the Reasoning Capability Tool component has been under development in our information assurance internal research and development project.

Conclusion

Development of a meta-language benefits both laymen and security engineers by allowing both to develop security policies in their own terms. The meta-language provides a translation mechanism between the end user's interpretation of security policy and the security practitioner's requirements for semantically sound security policy expressions. These policies, defined within the structures of the meta-language, can be stored in a repository for use by multiple external tools to determine compliance with the specified system security policy.

Acknowledgements

This research was conducted in part under a contract with the Defensive Information Warfare Branch of the U.S. Air Force Research Laboratory at Rome, NY (contract #F30602-00C-0057). Additional research has been conducted under Harris Corporation's Internal Research & Development program.

References

- [1] Jacob, Jeremy, "Security Specifications," Proceedings of the 1988 IEEE Symposium on Security and Privacy, Oakland, CA, 18-21 April 1988.
- [2] Dobson, John E., and McDermid, John A., "A Framework for Expressing Models of Security Policy," Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy, Oakland, CA, 1-3 May 1989
- [3] Tallet, Joseph, Ronda Henning and Kevin Fox, "IDEA: An Information Superstructure", to appear in the Proceedings of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX II), Anaheim, CA, 12-14 June 2001.
- [4] http://webopedia.internet.com/Computer_Science/meta.html
- [5] <http://web.comlab.ox.ac.uk/oucl/research/grants/b8.html>
- [6] <http://sorry.vse.cz/~tom/Linux-Announces/33.html>
- [7] <http://www.the-xml.com/>
- [8] Scambary, Joel, Stuart McClure, George Klutz, Hacking Exposed: Network Security Secrets & Solutions, McGraw-Hill, New York, 2001.

Appendix F

IDEA – An Information Superstructure

IDEA – An Information Superstructure

Joseph O. Tallet
jtallet@harris.com

Ronda R. Henning
rhenning@harris.com

Kevin L. Fox
Member IEEE
kfox@harris.com

Abstract

This past October, the Associated Press reported that for over a month, hackers effectively penetrated Microsoft's firewall and, potentially, had access to source code for future products. Over the past year, reported commercial espionage has been increasing steadily. These events underscore the necessity for security engineers to define and implement strong information assurance infrastructures. The concept of defense in depth is used to provide sufficient detection and defensive countermeasures to ensure an enterprise's information is protected. Over the past three years, Harris has conducted research leading to the development of an assurance superstructure tool: the Integrated Design Environment for Assurance (IDEA). Building upon concepts developed during the Network Vulnerability Tool (NVT) program, the IDEA program refined the design and provides a comprehensive assurance analysis superstructure. The IDEA program benefited from the NVT architecture to gain greater adaptive capabilities to address the evolving capabilities of external vulnerability assessment and risk management tools. This re-architected solution builds upon lessons learned during the NVT project.

Index terms — architecture, patterns, FuzzyFusion™, data fusion, fuzzy, information operations, information security.

1 INTRODUCTION / PURPOSE

For the last five years, Harris Corporation has been conducting research on a security engineering toolkit. The objective of our research was not to build yet another vulnerability scanner or risk assessment tool, but to provide reuse and a common operational picture among multiple tools. Our long-term vision is a toolkit that can be used by Security Engineers in the vulnerability and risk assessment of networked computer systems (Figure 1).

An important aspect of the development of a common operational picture is to make the results useful in the user's context. To that end, we determined that an assessment superstructure should accommodate the tool user's security policy, not simply the individual tools' interpretations of policy. This allows the user to tailor his analysis to reflect

unique enterprise policy elements. From the user perspective, the toolkit assists the Security Engineer in the assessment of a system relative to its security posture. This enables the engineer to answer the question, "Is there something about the system components that causes the policy to fail?"

Important capabilities and opportunities researched during the IDEA project collectively support security engineers. FuzzyFusion™ provides analysis of separate data sources combining analysis tools' results into a single vulnerability indicator. Meta-language for security policy definition promises singular policy support for both seasoned security engineers and small business seeking to define or improve security policy. The Common System Model (CSM) provides a generic standard for representing system networks. This standard allows individual vulnerability assessment tools to openly share data.

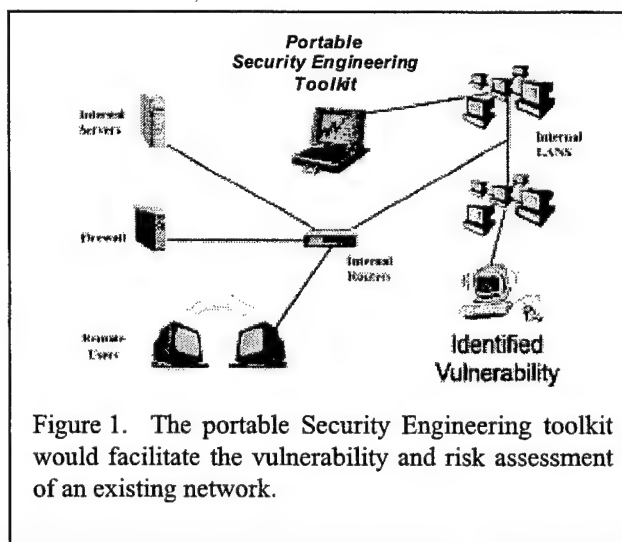


Figure 1. The portable Security Engineering toolkit would facilitate the vulnerability and risk assessment of an existing network.

IDEA incorporates all of these features as well as vulnerability metrics and measurement into an open architecture highly suitable for all aspects of security engineering and vulnerability assessment.

2 THE CONCEPT

From a conceptual perspective, the Security Engineer might have a variety of tools available – commercial off-the-shelf (COTS), government off-the-shelf (GOTS), shareware, etc. The engineer's tools might include the following.

- Network scanners – for network discovery of the supporting devices
- Vulnerability scanners – to find vulnerabilities in the OS, applications, etc.
- Risk analysis tools – addressing areas such as physical security, risk of data disclosure, valuation of data assets, etc.
- Threat analysis tools – examining courses of action an attack may take (e.g. shortest path).

For the security engineer to produce an accurate, system-level assessment of risk, multiple types of Information Assurance (IA) tools and techniques must be employed (Figure 1). The security engineer must become proficient in the use of each of these and the interpretation of their results. Among the categories of tools available to the security engineer are the following.

- Tools that work from documented vulnerability databases and possibly repair known vulnerabilities. Examples include ISS' Internet Scanner, Network Associates, Inc.'s CyberCop, and Harris' STAT™.
- Tools that use various parameters to calculate a risk indicator. An example is the Los Alamos Vulnerability Assessment (LAVA) tool.
- Tools that examine a particular aspect of the system, such as the operating system or database management system, but ignore the other system components. SATAN, for example, analyzes operating system vulnerabilities but ignores infrastructure components such as routers.

The use of *multiple* tools, from a variety of vendors, is a labor-intensive task. Typically, it means that the security engineer will have to enter a description or representation of a system (network) *multiple* times, in *multiple* formats. The security engineer then must manually analyze, consolidate and merge the resulting outputs from these *multiple* tools into a single report of a network's security posture. Afterwards, the security engineer can complete the risk analysis (calculating expected annual loss, surveying controls, etc.), and then repeat the whole process to analyze alternatives among security risk, system performance, mission functionality and the development budget.

No one particular risk assessment tool provides the complete coverage necessary for security engineers to ascertain a given system's vulnerabilities. It is a fact that complete vulnerability assessment requires system analysis using multiple vendors. Security engineers have an unfairly difficult task in assessing, defining, and implementing security profiles across

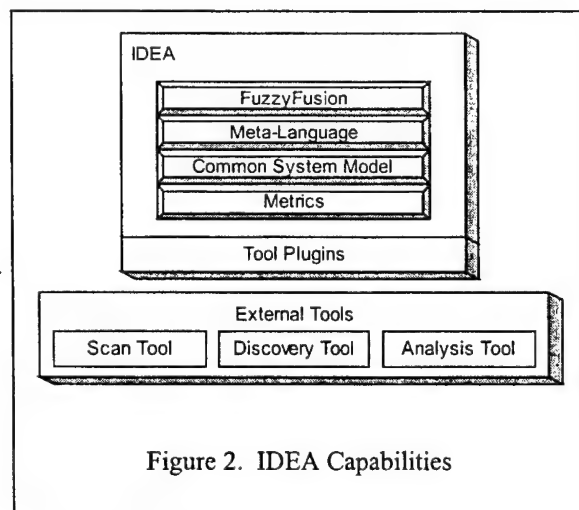


Figure 2. IDEA Capabilities

their networks. The ease of downloading and launching vulnerability assessment tools surpasses the effort necessary to correctly configure a corporation's firewall. The various design tools available to the security engineer do not, individually, provide complete views and assessments of a network.

IDEA bridges this information and analysis gap between individual tools. The IDEA solution is to provide:

STAT™ (System Test and Analysis Tool) is a trademark of Harris Corporation, Melbourne, Florida.

- A single interface for vulnerability assessment, providing a single interface to a variety of COTS vulnerability assessment tools;
- A data exchange mechanism, the Common System Model (CSM); and
- Tool plug-in capabilities wrapped in an Application Program Interface (API).

3 FUZZYFUSION™

In FY00, we addressed the problem of how to consolidate and merge the results from multiple Information Assurance assessment and analysis applications, while preserving the intent of each tool's analysis. During previous research, two technologies appeared most promising – Data Fusion and Fuzzy Logic. Data Fusion appeared to provide a conceptual framework for addressing the problem of merging results from multiple vulnerability assessment and risk analysis tools. Fuzzy Logic, particularly Fuzzy Expert Systems, appeared to provide a viable mechanism to implement the fusion concepts in the Information Assurance (or, its counterpart, Information Operations) domain. Fuzzy Expert Systems can use and assimilate knowledge from multiple sources.

As part of our research effort, we investigated technologies that would support our goal of consolidating and merging the results from multiple analysis applications. We examined a variety of current assessment products, including the inputs and outputs those products require. From these, Harris Corporation has developed the concept of *FuzzyFusion™* for Information Assurance Risk Analysis to consolidate and merge the results from multiple vulnerability assessment and risk analysis tools.

Intelligence Data Fusion is a multi-level, multi-disciplinary-based information process to yield the integration of information from multiple intelligence sources (and perhaps multiple intelligence disciplines) to produce the most specific and comprehensive unified data about an entity (its situation, capabilities, the threat it imposes) [3]. The purpose of Data Fusion is to provide the best possible tailored intelligence information to consumers based on the available inputs. Data Fusion technology has been applied, for example, aboard Airborne Warning And Control System

(AWACS) to the problem of tracking enemy military aircraft and assessing the situation and any threat.

FuzzyFusion™ combines Data Fusion and Fuzzy Logic in a unique way for the purpose of generating a simple vulnerability assessment. Information about the vulnerabilities of each system in a network are collected and assessed. The result of this assessment is a single fuzzy value for each of the nodes within the network. This fuzzy value relates to the node's assessed risk or vulnerability level. The validity of this assessment method is still under research.

Our objective is to apply FuzzyFusion™ to combine multiple types of data, from multiple sources, with other contextual information to form an integrated view of a networked system's assurance posture. In particular, our objective is to provide IDEA users with a simple expression of the *vulnerability posture* of a given system or system design. This allows the user to perform "what if" analysis for functionality, performance, and countermeasure trades, for the purpose of refining and improving the existing system or system design.

4 META-LANGUAGE

DARPA envisioned the development of a meta-language and toolkit in which designers and analysts would express the relationships, causality, vulnerabilities, threats, and objectives inherent in a system using this common meta-language. A meta-language supported by assurance tools will allow system engineers and laymen to describe security profiles useable by various assurance analysis tools.

We developed the meta-language by investigating how system engineers analyze a system. System engineers define the goal of the system, designs the system, its users, environment, and requirements and more. The meta-language supports all the steps a system engineer utilizes for system definition and analysis. To achieve this goal, the meta-language allows the identification of atomic assurance qualities each of which are mapped into more general qualities. System certification and accreditation engineers identify basic elements and actions defining "good assurance practices" and store them into a security policy defined within the constructs of the meta-language.

The security policy is stored in the IDEA repository where it is readily translated for external vulnerability assessment tools as well as for FuzzyFusion™ analysis. FuzzyFusion™ applies the policy when performing analysis to synthesize the results generated from the vulnerability assessment of external assessment tools.

The meta-language grammar allows definition of many security features, enabling definition of complete security policies.

- System accessibility – internal-only, external-only, external and internal,
- System protections – password, biometrics, passkey,
- System features – firewall, encryption, Discretionary Access Control, Mandatory Access Control, audit control, etc.,
- System contents – non-classified, classified, etc.,
- System locale – internet, local network, isolated,

The meta-language benefits both laymen and security engineers by allowing both to develop security policies in their own terms. A layman's front-end tool to the meta-language translates high-level terms into meta-language structures. A security engineer, capable of understanding security-jargon grammar has security-domain terms translated into meta-language constructs. This capability is a packaging approach for the meta-language. The meta-language is capable of supporting both high-level and explicit policy definitions through a supporting user interface. These policies, defined within the structures of the meta-language are stored within a database for use by IDEA, FuzzyFusion™, and external tools.

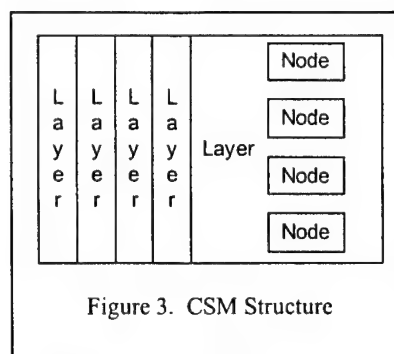
5 COMMON SYSTEM MODEL

The CSM is a data framework useful for representing external tool information within a common data format. It is a generic data framework for specifying networks, network components (computer and peripheral devices), and component features (operating system versions and patches, protocol capabilities). The CSM bridges the data

requirements of external tools allowing them to share common information.

System network information stored in the CSM is available to external tools as IDEA filters and formats it prior to launching external tools. IDEA actively collects, formats, and stores tool-generated data for later use.

The CSM structure is very generic (Figure 3). Layers representing system qualities may indicate network arrangements, wide area networks, local networks, etc. and contain nodes representing generic network items. A node may be a router, computer, switch, firewall, etc. and is identified by a unique name (i.e. URLs). Each node consists of attributes containing information collected by



system scan tools and utilized by system analysis tools for vulnerability assessment.

System engineers benefit from the CSM by providing system network information only once. IDEA handles the transfer of redundant network data among external tools.

5.1 Metrics

There has been an ongoing debate within the assurance community as to whether assurance characteristics are truly measurable. This discussion on information assurance metrics agrees that it is very difficult to define quantifiable assurance metrics, but that it may be possible to define assurance indicators, or tendencies, that would be less stringently defined. For example, the relative "goodness" of a system could be measured as opposed to an "assurance grade".

Given this lack of standard measures, providing some sort of assurance indicators required extensive

analysis. Over the course of IDEA, we defined and refined a set of ranking criteria. We eventually narrowed the assurance indicators to a standard set of base measures, or fundamental elements. These eight base measures and their meta-language expressions are:

1. Physical Access to system (Defeat Physical Security)
2. External Network Access (Defeat Firewall)
3. Access to a machine (Defeat I+A (Identification and Authentication))
4. Access to Segment (Defeat Network I+A (Identification and Authentication))
5. Access to Network (Defeat Routing)
6. Access to Sensitive Data (Defeat Access Control)
7. Access to Critical Applications (Defeat Access Control)
8. Access to Logs (Defeat Auditing)

All of these measures are related to *access* to the system. Over the course of our study we concluded that most assurance measurements were, in reality, probability calculations. That is, the assurance measurements were the probability that a given user could gain access to some system element whose access should have been denied by the defined system security policy.

6 METHOD

Using these fundamental concepts, one can envision an analogous Data Fusion process applied to the Information Assurance domain. In this model, the primary sensors are the various vulnerability assessment and risk analysis tools, complemented by various information solicited from the analyst through the graphical user interface. The resulting outputs from these tools are both qualitative and quantitative data, in a variety of formats generated by the various tool vendors. In the information assurance domain, the objects of interest are the nodes in a network or computing system – i.e. the assets, including hardware, software and data. The situation of interest is an assessment of the weaknesses in the assurance mechanisms of a computer network segment. These weaknesses may be exploited to cause harm or loss of confidentiality, integrity or availability. Assessing the risk faced by a computing system involves an assessment of the

possible threats¹, their likelihood of occurrence (exploitation), and the expected cost of the loss (or harm). Finally, the system can be refined based on the results of cost-benefits analysis. This requires information on protective measures (controls or countermeasures) appropriate for use to counter particular vulnerabilities, as well as the relative costs associated with deployment of these mechanisms. The cost-benefit analysis attempts to determine if the relative cost is less to implement a control or countermeasure, or to accept the expected costs associated with a breach of policy. This leads to the development of a security plan to improve the assurance posture of a computer network system. [1] [2]

7 IDEA ARCHITECTURE

The IDEA architecture is designed as an integration mechanism for a wide range of external security assessment tools and to be resilient to changes those

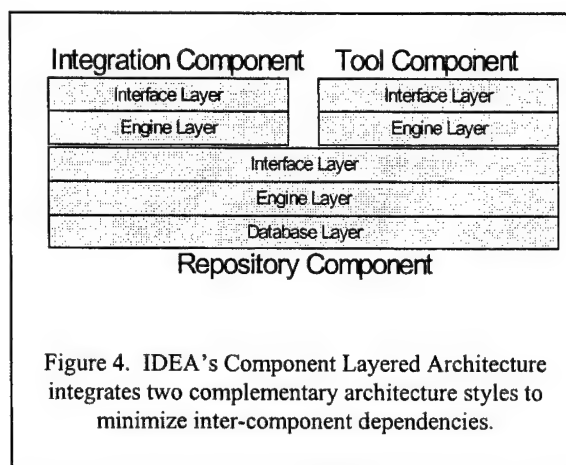


Figure 4. IDEA's Component Layered Architecture integrates two complementary architecture styles to minimize inter-component dependencies.

tools may impose. As such, IDEA interfaces to these tools through various integration layers designed into its Application Programming Interface (API). IDEA was designed with an open, layered API so it could be available to support as many tools as possible. In the IDEA architecture, each tool has a

¹ **Threats** to computing systems are circumstances that have the potential to cause loss or harm. Human attacks are examples of threats, as are natural disasters, inadvertent human errors, and internal hardware or software flaws. There are four kinds of threats to the security of a computing system: interruption, interception, modification and fabrication.

potential contribution to the overall assurance assessment of a given system. The IDEA user determines which tools are executed for a given assessment.

The IDEA design (Figure 4) combines two architectural styles: Components and Layers [5, 7]. The component style supports cohesion of behavior by grouping related assessment functionality into the components. The layer style supports loose coupling through standards enforced by the interface. Combining the component and layer styles, IDEA achieves a flexible, highly maintainable architecture.

The functionality of system vulnerability analysis is separated into three components. The layers illustrated in Figure 4 are realized through the design of each component.

- The Integration component – is responsible for interaction with external users and/or applications.
- The Tool component – provides direct interaction with assessment tools.
- The Repository component – provides a common data solution across tools and supports persistence for the other components.

7.1 Integration Component

The Integration component supports behaviors related to opening sessions, launching tools against a model, and viewing tool results. The intent of this design is to isolate management logic for session control within the IDEA Engine layer. The IDEA Interface layer is split into a GUI and API section. It is intended that all interactions with the IDEA Engine layer occur through calls to the API. The API provides all the necessary functions for external applications to perform vulnerability assessments on systems. That is, the Integration component is the portion of the superstructure that tools and analysts use to control tool execution.

The Integration component provides a complete set of assessment operations for security engineers. These operations support the discovery and design of network nodes, the collection of node configuration information, and the assessment of vulnerabilities as supported through specific external tools. Through the single interface of the Integration component, the security engineer is able to discover

an existing network or design a network and perform a comprehensive assessment of concealed vulnerabilities.

7.2 Tool Component

The Tool component contains the logic for the control of and interaction with external assessment tools. The dominant feature of this component is plug-in support for external tools. This support is achieved with the Maker pattern [6], a combination of the Abstract Factory and the Chain of Responsibility patterns [4].

Current assessment tools maintain their information within a file system structure or traditional database. The Tool component integrates with external tool storage capabilities, providing a data bridge across individual external tools. Any other assessment tool may use data collected by a given assessment tool. This data bridge is implemented within the Repository component. The Tool component extracts external tool information and ships it to the repository for storage. Information within the repository that may prove useful to other tools is collected by the Tool component.

The Tool component architecture supports rapid integration of external assessment tools. Integration impacts to the other components during integration of new tools are minimal. The Tool component supports a formal process to export tool capabilities across the Tool component boundaries. We chose this approach over the simple addition or modification of an external tool to minimize the impact on the external tools. Since IDEA has no control over these tools, we have no insight into their internal workings or alternative integration strategies.

Through the tool interface layer, the Integration component launches tools contained on the session's tool list. Each tool is responsible for its own repository data, and requests the model information required from the Repository Component. Upon completion of execution, each tool returns its respective results to the repository. A filter between the tool and the database converts tool output to a normalized form that is stored in the repository. This approach minimizes duplication of storage, and ensures that each tool receives its information in the format that it expects.

7.3 Repository Component

The Repository Component provides the database used by both the IDEA and Tool components. The design of the Repository component provides both a single interface into the database and a conversion layer to implement the logic required to convert data between the tools and repository's normalized Common System Model.

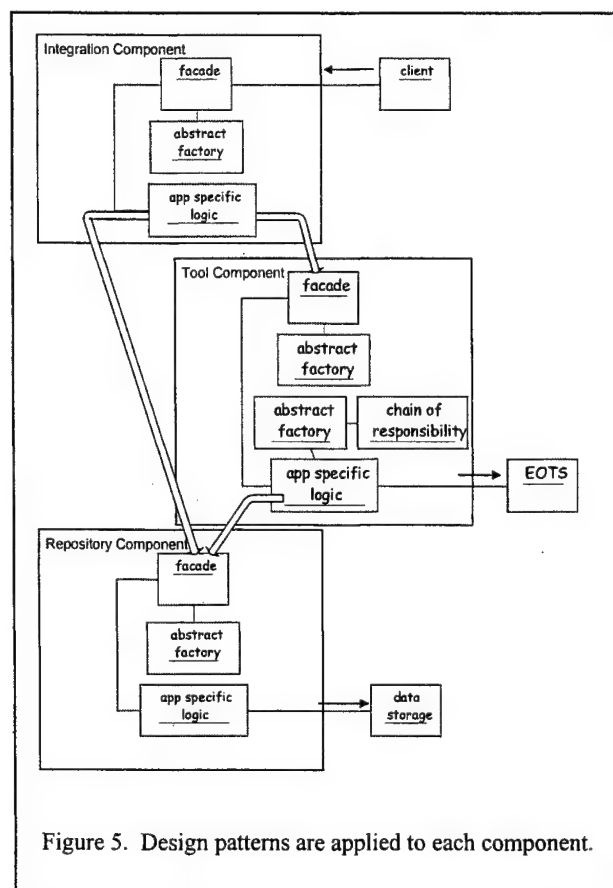


Figure 5. Design patterns are applied to each component.

8 THE ARCHITECTURE AT WORK

The Tool component and the Repository component work together to translate the results generated by each assessment tool into a common format. This format is stored within the CSM and is used by the Tool component as it launches various assessment tools. For example, discovery tools generate maps of networks they discover as their outputs. Scanning tools take this *discovered* information and the apply it, targeting network specific nodes/devices (routers,

etc.) and collecting feature information (OS version, etc.) about individual nodes. Once the attributes of the network nodes have been collected, this information is also stored in the CSM. The analysis tools examine the attributes of the network nodes in order to determine potential vulnerabilities. These are examples of how three categories of tools can share information among them. Each may use data collected by another tool to provide a complementary analysis. Each would, without the integration provided by IDEA, require security engineers to input redundant data for each tool, requiring additional user input and introducing the possibility for contradictory or incorrect information to be applied in the analysis. IDEA reduces the amount of effort necessary to successfully assess a network. IDEA can also, again via the CSM, detect when a tool may not have enough data to successfully perform its task. In this case, the security engineer is solicited for additional information that may be required. IDEA can reduce the amount of data entry required from the engineer by actively sharing data among vulnerability tools and, when more information is necessary, IDEA can actively collect that information through user interaction.

Various vulnerability assessment tools produce differing, sometimes inaccurate or conflicting results. Today, the security engineer determines which pieces of data are significant and which are of little consequence based on the expertise and interpretive skills of the tool user. That is, the repeatability of results depends upon the standardization of a given user's interpretation of a tool's results. IDEA incorporates FuzzyFusion™ logic to combine the results of risk assessments generated by individual vulnerability tools into a consolidated, repeatable response.

Using the security profile information stored within the Repository component, the consolidated response is generated by comparing tool results against the security profile data. Just as security policies differ among enterprises, security profiles differ from organization to organization as well. Security engineers are able to define and apply their own security profiles using IDEA's Meta-Language. These profiles are stored within the Repository and used by FuzzyFusion™ to correlate the vulnerability information collected. IDEA supports a basic set of requirements derived from the Trusted Computer System Evaluation Criteria (TCSEC), the Trusted

Network Interpretation of the TCSEC (TNI), and the Common Criteria. The engineer can tailor the basic set of requirements associated with a given security function to address the requirements associated with a given enterprise.

The Repository component is aware of the configuration of the various tools and how tool result data is to be stored in the CSM. Inserting a new tool into the IDEA framework consists of four easy steps:

1. Writing a driver for the tool.
2. Mapping the tool's input data to the CSM and security profile
3. Mapping the tool's output data to the CSM and security profile.
4. Defining the tool configuration information within the Repository component.

This is primarily a database-centric architecture. The repository is responsible for interpreting among tool data, CSM data, and security profile data.

The Tool component is used to pass this data to and from the external tools and the CSM. The Tool component is composed of several software patterns, enabling a complete data-level decoupling between the other two components and providing plug-in support for external tools. The decoupling is done with the Façade pattern. The plug-in is done with the Maker pattern.

The Integration component combines the capabilities of the Tool and Repository component, presenting the system engineer with a suite of capabilities for vulnerability assessment.

9 PATTERN APPLICATION

All three components are designed using a consistent application of a set of software design patterns (Figure 5) supporting loose coupling (minimizing cross-component dependencies), and tight cohesion (maximizing inter-component collaborations).

The Interface layer of each component is composed of a Façade and Abstract Factory pattern [4]. This set of patterns is used to enforce a decoupling between external clients and component-internal data structures. The Façade pattern provides external clients with a set of assessment capabilities. The Abstract Factory pattern is used to manage the

creation of component-internal objects to handle external requests.

The Integration component's Façade pattern supports network design and discovery by passing the operations on to the Tool component for direction. Network scanning and analysis are also passed to the Tool component for handling. Operations for viewing tool results are directed to the Repository component. The Integration component allows all external clients (GUI, scripts, external tools, etc.) access to vulnerability assessment and management operations.

The Tool component's Façade supports the launching of external tools. This design purposefully minimizes direct tool interaction from the Tool component's clients. The tool component uses the Maker pattern [6] to support the plug-in integration of external tools with minimum impact to existing code. Adding interfaces to external tools is as simple as adding a new class to handle the interaction with the Tool Component. Common calling techniques are enforced, ensuring that basic code modifications are uncomplicated and limited in scope.

The Repository component's Façade supports data storage and retrieval operations. This API is more flexible, allowing the Tool component to retrieve any necessary configuration and setup information for external tools. Tool result storage is also supported through this interface. At the backend of the Repository façade, a data interpreter ensures that tool data and CSM data are mapped to each other. The Integration component uses this interface to collect and store information for the security engineer.

10 CONCLUSIONS

The IDEA program provides the first information superstructure, an important support component for security design and assessment. The IDEA architecture facilitates the definition of security policies via a meta-language, provides a single point of entry to a potentially wide selection of security assessment tools, and incorporates FuzzyFusion™ analysis to generate a common operational picture for the security engineer. Bringing all these capabilities together in an open software architecture provides an invaluable tool for the security engineer. It also provides a collection of value-added functionality for external tool vendors wishing to

take advantage of its support for security policy engineering and/or FuzzyFusion™ analysis. FuzzyFusion™ provides evidential reasoning among disparate sources of vulnerability assessment information. The result: repeatable, rapid vulnerability assessments, enabling the security engineer to address system security issues efficiently. The eight base security measures identified during meta-language development are inherently useful for categorizing output results from FuzzyFusion™.

The meta-language standard for defining security policy information provides a useful basis for identification of policy building and policy enforcing tool suites.

11 RESULTS

Our research has resulted in a four-stage model of vulnerability fusion. This model is designed to reflect the global nature of a large enterprise, the increased dependence on inter-networks of information, and the vulnerabilities associated with individual network segments and devices. We have embodied the model in an information superstructure for vulnerability assessment tools, creating an integrated environment to provide the security engineer with a powerful tool suite. The model has been successfully applied to static vulnerability analysis, and has been integrated with current vulnerability assessment technologies for vulnerability identification and correlation.

The information superstructure can be applied to system designs to determine design vulnerabilities prior to implementation of a given system architecture. It can also be applied to legacy architectures to determine if the security posture of the system has been modified from the initial baseline. In either event, IDEA provides a repeatable set of results that can be used by an enterprise to provide a more comprehensive perspective on its security posture.

12 FUTURE WORK

The IDEA program leads the way in several vulnerability assessment technologies.

FuzzyFusion™, the process of consolidating and merging results of incongruent information sources has application outside the information assurance

arena. Harris is studying other domains for FuzzyFusion™ technology insertion. Harris is incorporating FuzzyFusion™ technology into its STAT® intrusion detection tool suite.

The security-centered meta-language builds inroads into collecting and documenting security profiles. Future work with the meta-language include design analysis tools which compare existing or hypothetical system networks with security profiles. Another avenue is automating the generation of security profiles given a system network.

The Common System Model provides the start of an information assurance data model standard. A common data format among assurance tools would be a boon to security engineers and greatly simplify vulnerability analysis.

While research into quantifiable assurance metrics continues, Harris has identified 8 supporting measures related to access. These measures offer a solid inroad into measuring system security. Our conclusion that standard probability calculations simplify the measurement of assurance allows us to focus on identifying high-probability access points to the system and point security engineers to security policy weaknesses and violations.

13 ACKNOWLEDGEMENTS

This research was conducted in part under a contract with the Defensive Information Warfare Branch of the U.S. Air Force Research Laboratory at Rome, NY (contract #F30602-00-C-0067). Additional research has been conducted under Harris Corporation's Internal Research & Development program.

14 REFERENCES

- [1] Charles P. Pfleeger, *Security in Computing*, Prentice Hall PTR, Upper Saddle River, NJ, 1997.
- [2] Deborah Russell and G. T. Gangemi, *Computer Security Basics*, O'Reilly & Associates, Inc. 1991.
- [3] Lofti A. Zadeh, "The Calculus of Fuzzy If/Then Rules", *AI Expert*, March 1992, Pp. 23-27.
- [4] Erich Gamma, [et al.], *Design Patterns Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, Inc., 1995.
- [5] Frank Buschmann, [et al.], *Pattern-Oriented Software Architecture*, John Wiley & Sons Ltd., 1998.
- [6] T. Culp, "Industrial Strength Pluggable Factories", *C++ Report*, 11(9), Oct. 1999.
- [7] M. Shaw, D. Garlan, *Software Architecture*, Prentice-Hall, Inc., Upper Saddle River, NJ, 1996.

ABOUT THE AUTHORS

Joseph O. Tallet received the B.S. degree in Computer Science from the Florida Institute of Technology, of Melbourne FL in 1988, the M.S. degree in Computer Science in 1994 from the Florida Institute of Technology.

He is the Chief Software Engineer and Software Architect for the IDEA Project. A member of Harris' Software Architecture Tools, Methods, and Technology Team, he actively participates in Harris' software process improvement, training, and is the reuse point of contact for Harris' Government Communication Systems Division (GCSD).

Ronda R. Henning received the B.A. from the University of Pittsburgh, the M.S. in Computer Science from Johns Hopkins University, and the M.B.A. from the Florida Institute of Technology. She is also a Certified Information Systems Security Professional (CISSP).

She is the senior Security Systems Engineer for Harris Corporation's Government Communications Systems Division. She currently leads the Information Assurance center of excellence, an interdisciplinary engineering group responsible for Information Assurance technology research and development, as well as assurance technology insertion on large-scale system integration opportunities. As a member of the Harris Engineering Process Group, she developed the Harris Secure Systems Engineering Guidebook. She was a founding member of the NSA/Industry consortium responsible for the System Security Engineering Capability Maturity Model (SSE-CMM). Prior to her employment at Harris, she was a deputy branch chief of information security research and development at NSA.

Kevin L. Fox received the B.S. degree in Mathematics from the University of the South, Sewanee, TN in 1981, the M.S. degree in

Mathematical Sciences from Clemson University in 1983, and the Ph.D. in Mathematical Sciences from Clemson University in 1987.

Since then, he has been a software engineer with Harris Corporation's Government Communications Systems Division. His research interests have spanned a variety of Information Technologies, including information retrieval, databases, natural language processing, neural networks and data fusion. Since 1998, he has been conducting research on tools to support Security Engineers in the assessment of existing networked systems for vulnerability and risks, as well as the design new networked systems to meet security requirements.

Dr. Fox is a member of the IEEE and the IEEE Computer Society.

***MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)***

*The advancement and application of Information Systems Science
and Technology to meet Air Force unique requirements for
Information Dominance and its transition to aerospace systems to
meet Air Force needs.*